2002-09

An agent-based architecture for generating interactive stories

Osborn, Brian A.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/9810

NAVAL POSTGRADUATE SCHOOL
Monterey, California

DISSERTATION

AN AGENT-BASED ARCHITECTURE FOR GENERATING INTERACTIVE STORIES

by

Brian A. Osborn

September 2002

Dissertation Supervisor: Michael Zyda

This dissertation was completed in cooperation with the MOVES Institute.

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2002	**3. REPORT TYPE AND DATES COVERED** Doctoral Dissertation

4. TITLE AND SUBTITLE: An Agent-Based Architecture for Generating Interactive Stories	**5. FUNDING NUMBERS**
6. AUTHOR Brian A. Osborn	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT *(maximum 200 words)*

The Department of Defense relies on modeling and simulation for a variety of purposes, including joint exercise training, developing and evaluating doctrine and tactics, and studying weapon system effectiveness. Advances in technology have made the achievement of technically and visually accurate simulations possible, but little has been done to present realistic scenarios while supporting user interaction. This dissertation describes a multi-agent interactive simulation engine for generating interactive scenarios or stories. A general-purpose multi-agent system simulation architecture, called a Connector-based Multi-Agent System (CMAS) is developed and presented, along with a software agent communication and coordination mechanism. In this architecture, stories are generated through discovery as a by-product of agent interactions, rather than being fixed in advance. The ensuing story adapts to the user's interventions and is closely aligned to the goals of the agents. The multi-agent system design of the story engine has resulted in a data-driven simulation engine, which is domain independent and highly scalable.

The story engine is fielded as the underlying simulation engine behind the U.S. Army's *America's Army: Soldiers* project. The instantiation of the story engine as it applies to *Soldiers* is presented. As a component of *Soldiers*, the story engine is an integral module in an interactive story generation system.

14. SUBJECT TERMS multi-agent system, multi-agent simulation, interactive stories, interactive narrative, interactive simulation, agents, scenario-based training	**15. NUMBER OF PAGES** 186
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release, distribution is unlimited.

AN AGENT-BASED ARCHITECTURE FOR GENERATING INTERACTIVE STORIES

Brian A. Osborn
Commander, United States Navy
B.A., University of Maine at Orono, 1982
M.S., Naval Postgraduate School, 1993

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2002

Author: ______________________________
Brian A. Osborn

Approved by:

Mike Zyda Professor of Computer Science Dissertation Supervisor	Ted Lewis Professor of Computer Science
Don Brutzman Associate Professor of Applied Science	Rudy Darken Associate Professor of Computer Science
Michael Capps Research Assistant Professor of Computer Science	John Hiles Research Professor of Computer Science

Approved by: ______________________________
Chris Eagle, Chair, Department of Computer Science

Approved by: ______________________________
Carson K. Eoyang, Associate Provost for Academic Affairs

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The Department of Defense relies on modeling and simulation for a variety of purposes, including joint exercise training, developing and evaluating doctrine and tactics, and studying weapon system effectiveness. Advances in technology have made the achievement of technically and visually accurate simulations possible, but little has been done to present realistic scenarios while supporting user interaction. This dissertation describes a multi-agent interactive simulation engine for generating interactive scenarios or stories. A general-purpose multi-agent system simulation architecture, called a Connector-based Multi-Agent System (CMAS) is developed and presented, along with a software agent communication and coordination mechanism. In this architecture, stories are generated through discovery as a by-product of agent interactions, rather than being fixed in advance. The ensuing story adapts to the user's interventions and is closely aligned to the goals of the agents. The multi-agent system design of the story engine has resulted in a data-driven simulation engine, which is domain independent and highly scalable.

The story engine is fielded as the underlying simulation engine behind the U.S. Army's *America's Army: Soldiers* project. The instantiation of the story engine as it applies to *Soldiers* is presented. As a component of *Soldiers*, the story engine is an integral module in an interactive story generation system.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF EQUATIONS

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

As always in an endeavor such as this, a number of people played a role in seeing the work from inception to conclusion. First I must recognize John Hiles—this dissertation would have been impossible without his guidance and inspiration, for which I am deeply grateful. His insights were crucial. I owe special gratitude as well to Mike Zyda. A true visionary, his recommendation led me down this path of interactive narrative. Ted Lewis deserves my thanks for his patience as I put thoughts on paper. Always a constructive listener, this document benefited greatly from his discerning feedback, and I benefited from our many hours of discussion. To Mike Capps, Rudy Darken, and Don Brutzman, my appreciation for the direction provided and experience shared.

This work progressed under the ready support of faculty, staff, and students of the Computer Science Department and MOVES Institute, with a special thanks to Margaret Davis for her assistance with the graphics that appear here. I wish to recognize my fellow doctoral students, who served as a well of ideas and support: Curt Blais, Simon Goerger, Perry McDowell, Mike VanPutte, and Joerg Wellbrink.

The US Army's Office of Economic and Manpower Assessment (OEMA) provided the inspiration behind the *America's Army* project. I am indebted to Lieutenant Colonel Casey Wardynski and the rest of the OEMA team for their help and belief in the project. The *America's Army: Soldiers* development team of Luke Ahearn, Mike Bailey, Neal Elzenga, Randy Jones, James Marsh, and John Mason is an inspired and dedicated team with a vision of producing a radically new genre of game; it was a privilege working with them.

To my family—Carrie, Chris, Melissa, and Alan—it is hopeless to try to express the depth of my gratitude for your love and encouragement. This dissertation is dedicated to you.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

"I like a good story well told. That is the reason I am sometimes forced to tell them myself."

Mark Twain, 1907

A. THESIS STATEMENT

Highly dynamic, interactive stories can be generated by a multi-agent system as a by-product of agent interactions. These stories are sensitive to user input, the actions, goals and internal states of autonomous characters, and the global state of the environment. By formulating the multi-agent system as a data-driven architecture, a domain independent story system can be developed that generates logically connected, goal-directed stories that are sound with respect to the domain of interest.

B. MOTIVATION

In 1997 the National Research Council (NRC) issued a report that specified a joint research agenda for defense and entertainment modeling and simulation [NRC, 1997]. The NRC report provides a guide as to what research and development needs to be done to develop future interactive entertainment and defense modeling and simulation systems. For the entertainment industry, as stated in the NRC report, modeling and simulation technology lies at the heart of video games, theme park attractions, and entertainment centers. The Department of Defense (DoD) uses modeling and simulation for a variety of purposes, such as to conduct joint training exercises, develop and evaluate new doctrine and tactics, analyze alternative force structures, and study the effectiveness of new weapons systems. Advances in information technology have lowered the cost of computer-based models and simulation, making modeling and simulation a cost-effective alternative to live training and exercises.

The NRC report identified five areas as having potential for greater collaboration between the entertainment industry and defense, one of which included computer-generated characters. It is in this area, specifically the control of such characters within the bounds of an interactive story, that this research focuses.

C. PROBLEM STATEMENT

An interactive story system should support a complicated story world comprised of characters and props, along with locations where events occur and characters interact with each other, the props, and their surroundings. The interactions should not be random, nor should the characters be free to act as they wish without regard to the rest of the story world. There are constraints on what can be done and how interactions occur. At the same time, it must be adaptable to multiple domains, whether it be presenting training scenarios in a ground combat simulation or an action-adventure story. To make this story system interactive, allowances must be made for a user to intervene as the story progresses, while at the same time ensuring the story remains logically connected and aligned with the protagonist's goals.

Current approaches based on artificial intelligence top-down planning techniques can support complicated plots with a diverse set of story characters, but they are extremely domain-knowledge specific. All possible character-to-character and character-

to-environment interactions must be defined in advance along with a predefined story script or plan. Extensive time and effort is required to generate knowledge bases and dependency networks for each new story [Schank and Abelson, 1977], [Young, 1999]. Algorithmic approaches using tree or graph structures to organize story events provide a domain independent methodology, but for complicated stories, these approaches are overcome by the combinatorial problem of evaluating all possible plots each time an event occurs [Weyhrauch, 1997]. Similarly, if the story is highly scripted and the corresponding graph structure simplified, the resulting stories are akin to highly immersive hypertext documents where the branch points are tightly constrained in order to meet pedagogical goals or conform closely to the author's story line [Marsella *et al*., 2000], [Swartout *et al*., 2001].

Thus, previous approaches have taken a largely top-down view of creating a story, where the author must in some way anticipate all actions the interactive user might take at each step of the story. The problem of creating a general interactive story system is one of developing an architecture that scales well to large story worlds where all possible interactions may not be known a priori, while at the same time remaining domain independent.

1. Definition of Interactive Story

The following definition of interactive story is taken from Brenda Laurel's dissertation *Toward the Design of a Computer-Based Interactive Fantasy System*.

> An interactive story is a first-person experience within a fantasy world, in which the user may create, enact, and observe a character whose choices and actions affect the course of events just as they might in a play. The structure of the system ... enables first-person participation of the user in the development of the story or plot, and orchestrates system-controlled events and characters so as to move the action forward in a dramatically interesting way [Laurel, 1986].

Her definition describes very well what an interactive story experience should be like and is a starting point for designing an interactive story system. Laurel states that the system must "move the action forward in a dramatically interesting way." This implies the story is not being driven to a single predetermined ending; rather the climax and ending are

free to change as the story progresses. The system strives to present the best story that emerges based on the user interaction, as opposed to presenting a specific story.

2. Interactive Story

For the purposes of discussion, it is assumed that a fully defined and functional "fantasy world" exists. The existence of this world implies that there are laws and rules in place to govern the interactions and events in this world. The world is populated with autonomous computer controlled characters, driven by internal goals and cognizant of the laws and rules of the world. The autonomous characters are free to explore and manipulate the environment, again subject to the rules of the world. The interactive user is a participating character in the world. Like the autonomous characters, the user's character is free to roam and interact in the story world. The difference is that the goals, desires and actions of the user's character are influenced not only by the environment and interactions with the other characters, but also directly by the user.

The interaction of the user impacts the development of the story, but not at discrete points in an obtrusive way. Rather, the user's interaction serves to mold the plot in a continuous fashion. Michael Mateas and Andrew Stern describe the evolution of an interactive story plot in their paper *Towards Integrating Plot and Character for Interactive Drama*. They write:

> Changes in the plot should not be traceable to distinct branch points; the player will not be offered an occasional small number of obvious choices that force the plot in a different direction. Rather, the plot should be smoothly mutable, varying in response to some global state which is itself a function of the many small actions performed by the player throughout the experience [Mateas and Stern, 2000].

As the user interacts with the story world, the events and characters in the world are influenced so as to move the action forward in an interesting way. The influence can be as subtle as a slight adjustment to a character's personality to encourage a certain type of behavior, or as overt as overriding their goals to force an action that is in the best interest of the overall story. Obviously the subtle approach is preferred.

The challenges of presenting a compelling story in the face of interactivity are well documented [Bates, 1992], [Murray, 1998], [Weyhrauch, 1997], [Mateas and Stern 2000], [Szilas, 2001]. A good story is well planned and scripted. The author goes to

great lengths to ensure events unfold in a sequence that best captures the author's desires and is interesting to the audience. In the case of an interactive story, the task of writing and presenting the story is more complicated because the audience is one of the characters in the story (the interactive user). In which case, the author must forfeit some measure of control over how the story plays out in exchange for interactivity. This balance between author control and interactive freedom for the user is at the heart of developing an interactive story world capable of producing a large number of diverse story lines.

In the model of interactive story presented in this dissertation, the author exerts a high level of control over developing the story world, but forfeits *direct* control over the events that occur (Figure 1). The story world is defined through laws and rules that govern character-to-character and character-to-environment interactions. Included in this definition of the world are the autonomous story characters with their own goals and personalities. While direct control over the events is limited, the temporal relationship between events is established indirectly through the laws and rules imposed by the author on the story world.

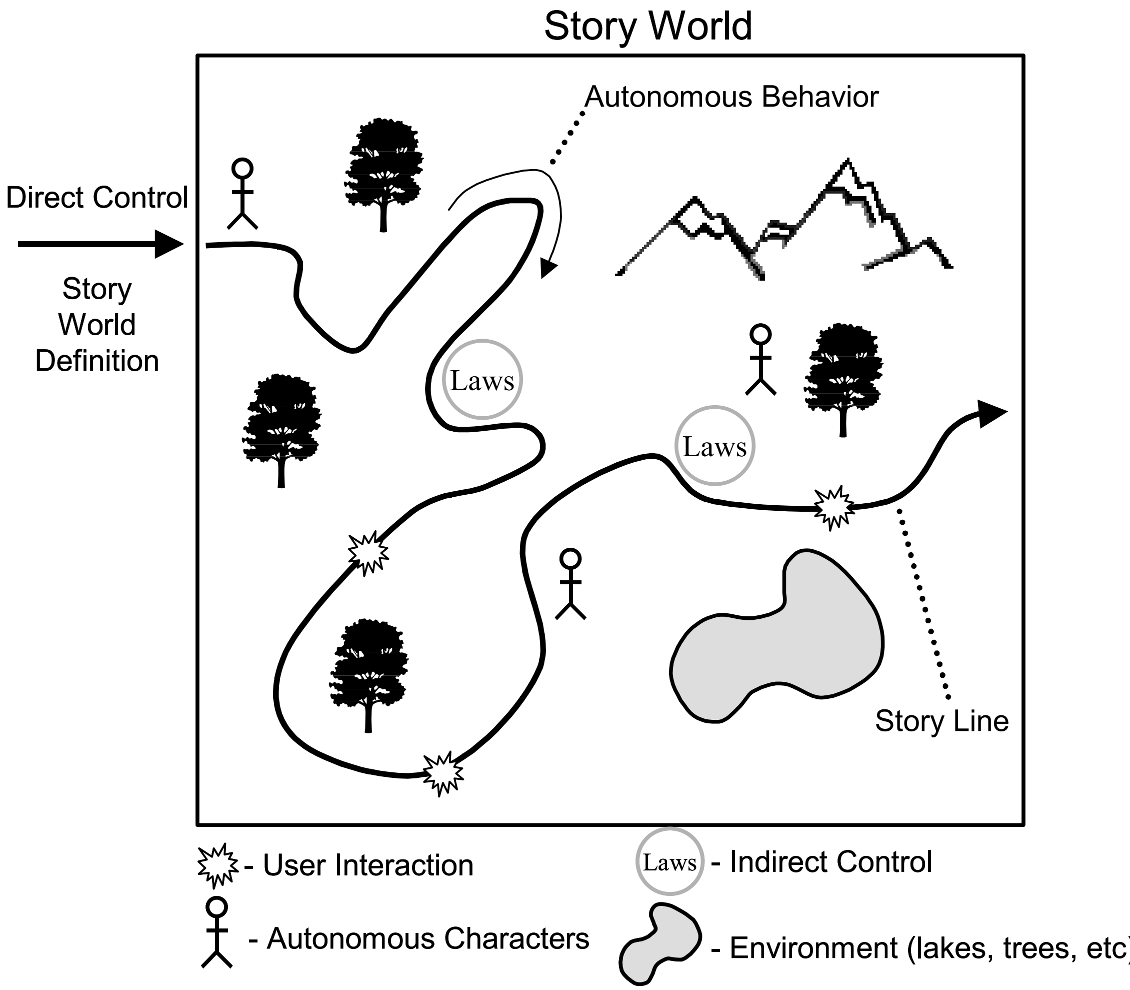


Figure 1. Story World Diagram

The story characters, including the character being influenced by the user, are turned loose in the story world. The result is a story line that results from the interaction of the characters with each other, the environment and the user. It is conceivable that a single story world could yield many different story lines.

The character's goals serve to focus the interactions so as to produce an "interesting" story, where "interesting" is relative to the story world domain and the objectives of the author. If the system is designed for war-game scenarios, "interesting" may be defined in terms of the feasibility and uniqueness of the scenarios and how well they meet the training requirements. If the desire is to present an action-adventure story, "interesting" may be defined in more classical narrative terms employed by screenplay writers and authors of fictional works.

3. Summary

Interactive story concerns itself with building simulated worlds populated with autonomous characters and entities, within which the user participates in the story or simulation. A significant amount of research has taken place in the area of building believable autonomous characters to populate the story world [Reilly, 1996], [Loyall 1997], [Mateas, 1997], [Blumberg, 1996], [Hayes-Roth *et al.*, 1996] and [Rickel *et al.*, 2001]. However, the body of work on developing systems where the story lines and scenarios adjust to the autonomous behavior of individual agents is smaller in scope [Weyhrauch, 1997], [Marsella *et al.*, 2000], [Mateas and Stern, 2000] and [Swartout *et al.*, 2001].

The goal of interactive simulation, whether it is a virtual story or a combat simulation, is to present the user with an experience that suspends their disbelief in the artificialities imposed by the system. In this way, the user feels it is a "real" experience. From the DoD perspective, this results in more realistic and effective training, as well as more accurate assessments of the systems, tactics or doctrine being evaluated.

The entertainment industry has long known that to achieve this suspension of disbelief, it is not sufficient to simply produce a technically accurate simulation. It is the unfolding of events and presentation of the story, along with rich believable characters, that makes for a truly effective and immersive experience.

D. APPROACH

This research formally describes the architecture of MAISE (Multi-Agent Interactive Story Engine)[1], a simulation engine that uses a multi-agent system (MAS) to produce interactive stories. A domain specific story world is constructed from domain independent agent-based modeling constructs. The constructs of *tickets* and *connectors* [Hiles *et al.*, 2001] are combined with *scenes* and *interactions* to construct dynamic plans that manifest themselves as goal-directed stories. The intention is not to tell a specific story, but to let the story unfold within the bounds of the story world based on the agents' interactions with one another and the user's interaction with the system. A typical story consists of goal driven autonomous characters, props, constraints, and a collection of

[1] For the remainder of the dissertation, MAISE is referred to simply as the story engine.

potential scenes, along with media, dialog, and character interactions to populate the scenes. These story elements are combined dynamically at run-time to generate a story line that adapts to the user's interventions and is closely aligned to the goals of the protagonist.

In this architecture, story lines are a by-product of observing constraints defined on classes of agents within the simulation. Thus, story lines are plans generated through discovery rather than following fixed plans defined beforehand. The story lines are generated through a simulation process called *connecting*, whereby agents are bound together according to a "best-fit" axiom. The by-product of the *connection* is the next scene in the story (or step in the plan). In this way, dynamic story lines, or plans, are evolved as the simulation runs. The bottom-up, MAS design of the story engine has resulted in a simulation engine that is domain independent and can be scaled to accommodate stories of any breadth and depth.

The story engine is an instance of a larger family of simulations entitled Connector-based Multi-Agent System simulations (CMAS). This dissertation formally presents the CMAS architecture, followed by the specific description as it applies to the story engine.

Finally, the story engine was fielded as the underlying simulation engine behind the U. S. Army's *America's Army: Soldiers* (*AA: Soldiers*) project. The instantiation of the story engine, as it applies to *AA: Soldiers*, is presented. As a component of *AA: Soldiers*, the story engine is an integral module in an Interactive Story Generation System (ISGS) that is patterned on the model-view-controller architecture (Figure 2). The story engine encompasses the model and controller, while the Scene Rendering Subsystem provides the view. Interactive access to the story world, via the story engine, is provided by the graphical interface. Development of the Scene Rendering Subsystem and graphical interface is outside the scope of this dissertation, however, they are described briefly in Chapter VI.

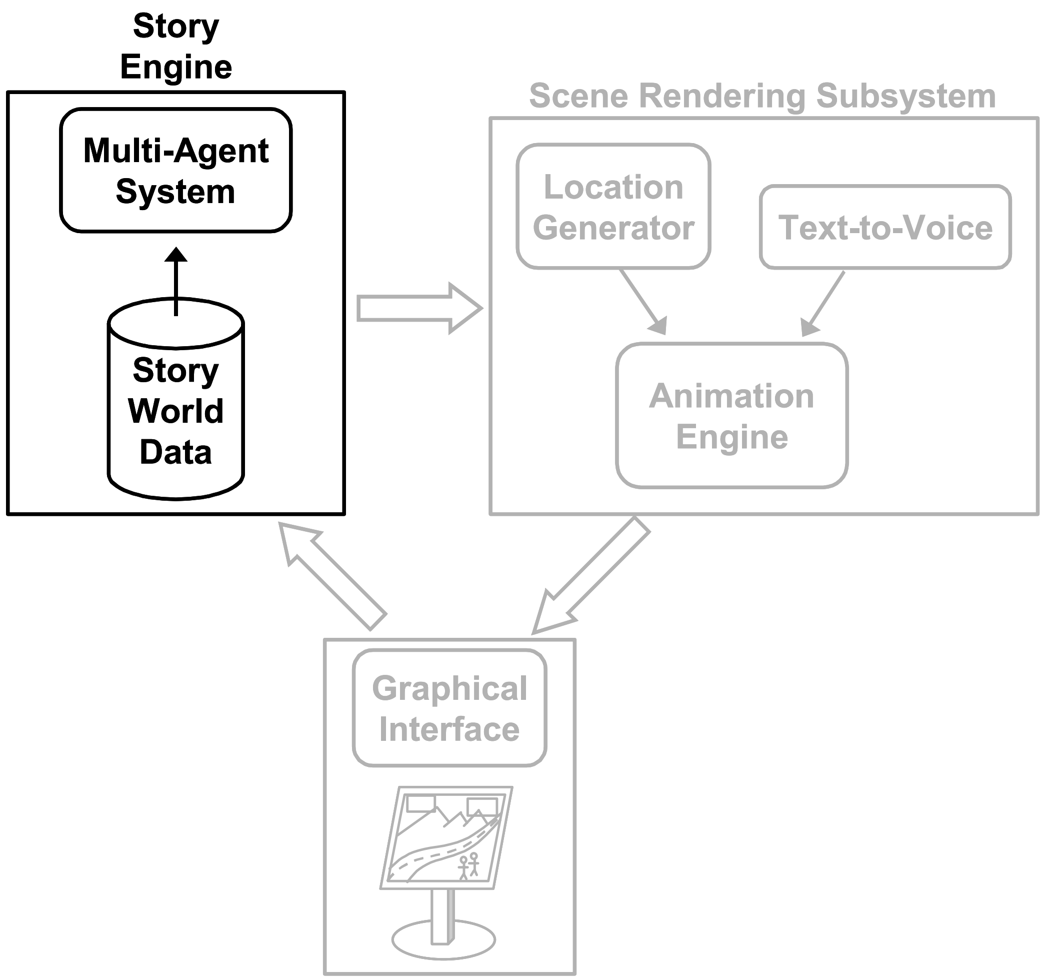


Figure 2. Interactive Story Generation System

E. CONTRIBUTIONS

This dissertation provides a fundamental new approach to generating interactive stories. The following are the specific contributions of this research:

- A formal description of a general-purpose multi-agent system architecture, called a Connector-based Multi-Agent System (CMAS).
- A domain independent, scalable simulation architecture, based on the CMAS architecture, for generating logically connected and goal-directed interactive stories (Story Engine CMAS).
- A bottom-up simulation methodology to discover novel and unexpected plans for achieving an agent's goal(s).

- A formal description of an agent communication, coordination and control process based on *connectors* and *connecting*.

F. DISSERTATION ORGANIZATION

The remainder of this dissertation is organized as follows:

- **Chapter II. Related Work.** This chapter is provided to give the reader a background on research conducted in the area of interactive narrative and multi-agent simulation planning systems. The work that is most relevant to this dissertation is described in detail.
- **Chapter III. Multi-Agent Systems.** Multi-agent systems are defined and a working framework describing multi-agent systems is presented. This framework is based on work by Jaques Ferber [Ferber, 1999].
- **Chapter IV. Multi-Agent System Research.** A survey of MOVES Institute computer generated autonomy research is presented along with a formal definition of *connectors* and *tickets*. The research presented in this chapter is fundamental to the development of the interactive story engine.
- **Chapter V. Connector-based Multi-Agent System Architecture.** This chapter brings together tickets, connectors and multi-agent systems to formulate a formal definition of a Connector-based Multi-Agent System (CMAS) architecture. A simulation model is developed based on the process of *connecting* and the agent interaction that occurs during a *connection*.
- **Chapter VI. Interactive Story Generation System.** The architecture of the story engine is presented in terms of the formal description of a Connector-based Multi-Agent System (CMAS). The story engine is combined with a real-time movie animator and generative text-to-voice system to create an Interactive Story Generation System (ISGS).
- **Chapter VII. CMAS Data Set Implementation - America's Army: Soldiers.** *America's Army* is a suite of two PC based computer games developed for the U.S. Army intended to provide young people with accurate, easy-to-assimilate information about the Army. *America's Army: Soldiers* is a story-based role-playing game where a player guides a character through an

Army career. This section describes the tickets, connectors, scenes and characters, including their goals and personalities used to create the story world. *America's Army: Soldiers* is presented as a proof-of-concept of this dissertation research.

- **Chapter VIII. Conclusions and Recommendations.** The dissertation concludes with a summary of contributions and suggested applications for the story engine. A number of avenues for follow-on work and extensions are also provided.

THIS PAGE INTENTIONALLY LEFT BLANK

II. RELATED WORK

A. INTRODUCTION

Stories have long played a central role in human culture. Learning, communication, social interaction, arts and even recreation all center around stories. Before man could write, stories were used as a means of passing information from one generation to the next. It was much easier for scholars to pass along important findings and key events in the culture's history when they were organized and relayed as a story. Along with being used as a means of exchanging information, stories provide a framework for how we approach the world. They provide meaning to the reams of data that flood our senses every day. Through the use of stories, whether consciously or sub-consciously, we make sense of the world. We order its events and find meaning in them by assimilating them into familiar narratives.

The media for relating stories has traditionally taken on familiar forms such as written work (rock wall carvings, ancient scrolls, books, magazines and newspapers), live performance (story-telling, theatrical plays and musicals) and cinema. These forms all have one thing in common; they are intended to tell a story to a non-participating audience. The stories are not written with the intention of changing the plot as the story progresses based on the desires of the audience. From the moment the first scene opens, every act, every action, and every line of dialog is scripted to achieve the specific goals of the author. There are no mechanisms for the audience to influence the story as it is being told. Why not? Because it is very difficult from the literary perspective. How does one tell or write a good story if they don't know what the characters are going to do or how they are going to react, particularly when it is one of the main characters (the interactive user).

This chapter examines alternative approaches that have been used in the interactive drama domain and related domains. The evolution of interactive entertainment and story understanding and telling systems are also presented from the perspective of their influence on current approaches to interactive narrative. Along the way, those efforts that best define the current state of research in the field of interactive story generation have been expanded upon. Finally, this chapter lays the groundwork for

a multi-agent system simulation approach to generating stories by examining two agent-based planning systems.

B. NARRATIVE INTELLIGENCE

Narrative Intelligence is a term that was coined by Marc Davis and Michael Travers of MIT's Media Lab in 1990 to describe a field of study combining artificial intelligence and literary theory [Davis and Travers, 1997]. Their collaboration was the foundation of a group of students and faculty interested in pursuing interdisciplinary work at the intersection of artificial intelligence, literary theory, media studies, cognitive science and human-computer interaction design. Drawing on a diverse range of influences, the researchers have explored a wide variety of topics relevant to narrative intelligence. A number of common themes have emerged including; narrative as the basis for human-computer interface design [Laurel, 1991], intelligent agents using narrative structure to model aspects of human intelligence [Schank, 1990], story-understanding systems, story-telling systems, interactive entertainment, and interactive drama.

While the term Narrative Intelligence emerged in 1990, early foundations of this area of study are grounded in the field of artificial intelligence (AI).

1. Artificial Intelligence Story Systems

In the 1970's and early 1980's, there was a substantial amount of interest in story understanding and generation. A research group at Yale, headed by Roger Schank, explored the use of scripts, plans and goals to understand narratives [Schank and Abelson, 1977]. Scripts and plans are a means for achieving goals. Scripts are used to capture the notion of a stereotyped situation and the sequence of actions appropriate for the situation. They are prepackaged sets of expectations, inferences, and knowledge that are applied in common situations. For example, in a restaurant script, the sequence of actions of entering, finding a seat, ordering a meal from a server, the meal being prepared and served, eating, paying the bill and leaving the restaurant would constitute a script. Scripts provide rules for understanding the connectivity in a stereotypical situation.

Since it is not possible to define a script for every possible situation, the notion of a "plan" was developed to deal with situations not previously encountered. Plans are a

repository of general information and provide a means of connecting events that cannot be connected by the use of an available script. They provide the mechanism for understanding events about which there is no specific information.

The Yale group, as part of their work, developed a series of programs to help understand textual narratives. These programs included SAM (Script Applier Mechanism), TAIL-SPIN [Meehan, 1976], and PAM (Plan Applier Mechanism). These early systems were intensely knowledge based, functioning in very limited domains. To make them more general, extensive knowledge engineering would be required. In addition, the plans were static; bound before the story ever began. Discussion of the specifics of each program is presented in the following sections on story-understanding and story-telling.

2. Story-Understanding Systems

Story-understanding systems are systems designed to take as input a narrative discourse, and provide as output information about the story. The information is normally shared as part of a two-way question and answer dialog between the user and the software program. The goal is for the program to not only recite facts directly from the story, but also be able to provide the user with information that could be logically inferred from the story events.

SAM was a program written at Yale designed to understand stories that rely heavily on scripts. To understand the stories, SAM created a linked causal chain of what took place in each story. A script applier then made inferences about events that must have occurred between events specifically mentioned. This resulted in a large "conceptual dependency network." From this, the system was able to paraphrase the original story, expand the story with the inferred events, or summarize the story based on measures of the relative importance of events. The network could also be queried to answer questions about the story.

PAM was another story-understanding system, but unlike SAM, which was based on scripts, PAM was based on plans. PAM used knowledge about goals and plans to discern the intentions of the characters in the story. The program kept track of the goals of each character and interpreted their actions as means of achieving those goals.

3. Story-Telling Systems

Unlike story-understanding programs, story-telling systems make no attempt at understanding the motivation behind the events and actions of the characters nor making inferences about why the events occurred. TALE-SPIN is an example of an early story-telling program written by Jim Meehan while a student at Yale in the mid-1970's [Meehan, 1976]. TALE-SPIN made up non-interactive stories about animals by simulating a world, assigning goals to the characters, and describing what occurred when these characters interacted with other characters and objects in the environment in the pursuit of their goals. It was used as a means of testing the goal and planning apparatus proposed by Schank's group. TALE-SPIN worked by first defining, in great detail, how a character might plan to accomplish a goal. The system required explicit plans for every possible goal a character might wish to achieve. Then, given a set of characters and a starting goal, the program could generate stories using its basic knowledge of the elements needed to construct a story, the elements of planning and achieving goals, knowledge of language structure, as well as general story world knowledge.

AI planning techniques are still being explored today as models for narrative plots. Michael Young, lead researcher at North Carolina State University's Liquid Narrative Group is extending recent techniques in AI planning, including causal and hierarchical structures of plans, to capture the key features of narrative structure [Young, 1999], [Young, 2000].

C. NARRATIVE THEORY AND SCREENPLAY STRUCTURES

The breadth and depth of literature on narrative theory, dating back to 330 BC [Aristotle, 330 BC], is sufficiently exhaustive so as to prevent a thorough review here. However, a brief, high-level description of story structure is presented.

Story structure is often described as a dramatic arc where the vertical axis represents tension and the horizontal axis is time (Figure 3). As the story progresses, tension builds and falls based on incidents and interactions in the story. All the while the overall slope of the arc shows an increase in tension. This continues until some crisis occurs resulting in the climax. During the climax, questions are answered and the tensions are resolved.

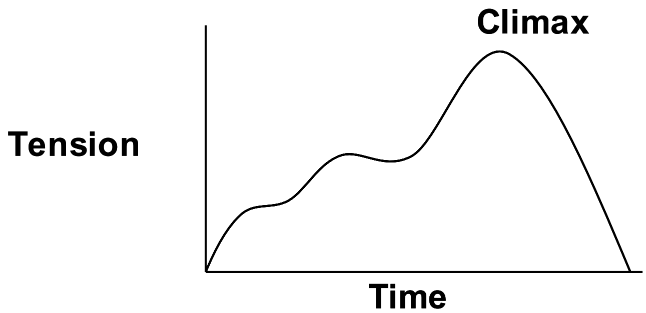


Figure 3. Dramatic Structure

Recent examinations of the movie industry and screenplay writing have produced a variety of multi-act structural models for representing dramatic arc. These models fit well with the task of generating interactive stories, where the user's view is more closely aligned with interacting in a movie or play than it is with reading a novel.

In David Siegel's essay *The Nine-Act Structure* [Siegel, 2001], he describes a general model for a movie's structure based on either seven or nine acts (Figure 4). His comparative analysis between his model and a number of recent movies indicates that a high percentage of top grossing films have followed the nine-act structure.

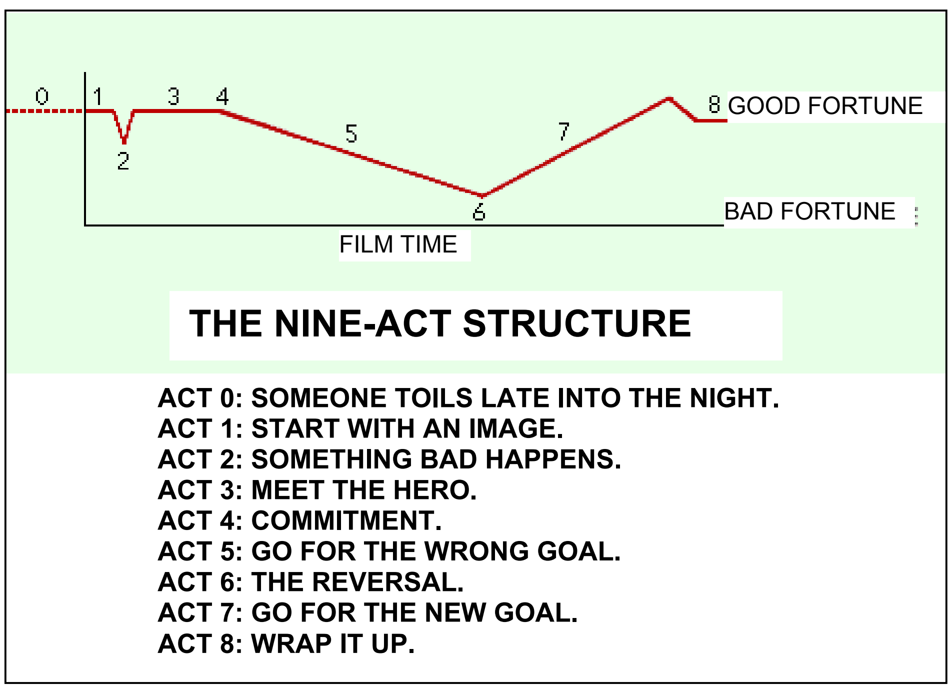


Figure 4. David Siegel's Nine-Act Structure from [Siegel, 2001]

At a more coarse level of granularity, Kristin Thompson presents a four-act structure for movie making [Thompson, 1999]. The acts include setup, complicating action, development, and climax. In setup, the initial situation is presented. This is often where the protagonist establishes one or more goals. The complicating action takes the action in a new direction. This may involve the protagonist pursuing a previously established goal in a new way, or possibly being faced with an entirely new goal or obstacle. Development is recognized as the phase of the story where the protagonist works toward the achievement of their primary goal(s). As the climax portion begins, the action shifts to straightforward progress toward a final resolution.

Stuart Voytilla cites 50 major films produced over the past 70 years as evidence of Christopher Vogler's movie storytelling model based on the protagonist's journey through 12 stages of a story [Voytilla, 1999]. Metaphorically speaking, the 12 stages take the protagonist from their everyday life, to the acceptance of some challenge, through the struggles for achievement and finally back home. Vogler's "Hero's Journey Model" is depicted in Figure 5.

In spite of their apparent differences, each of these models is functionally equivalent, with the primary difference being the level of fidelity of the model. The twelve phases of the hero's journey relate directly to the four acts identified by Thompson. Likewise, Siegel's nine-act structure can be mapped onto a four-act structure. Additionally, these models share a common link between the stages of the story and the central character's goals. In each case, transitions from one stage to the next are initiated by a goal shift on the part of the protagonist or another key character.

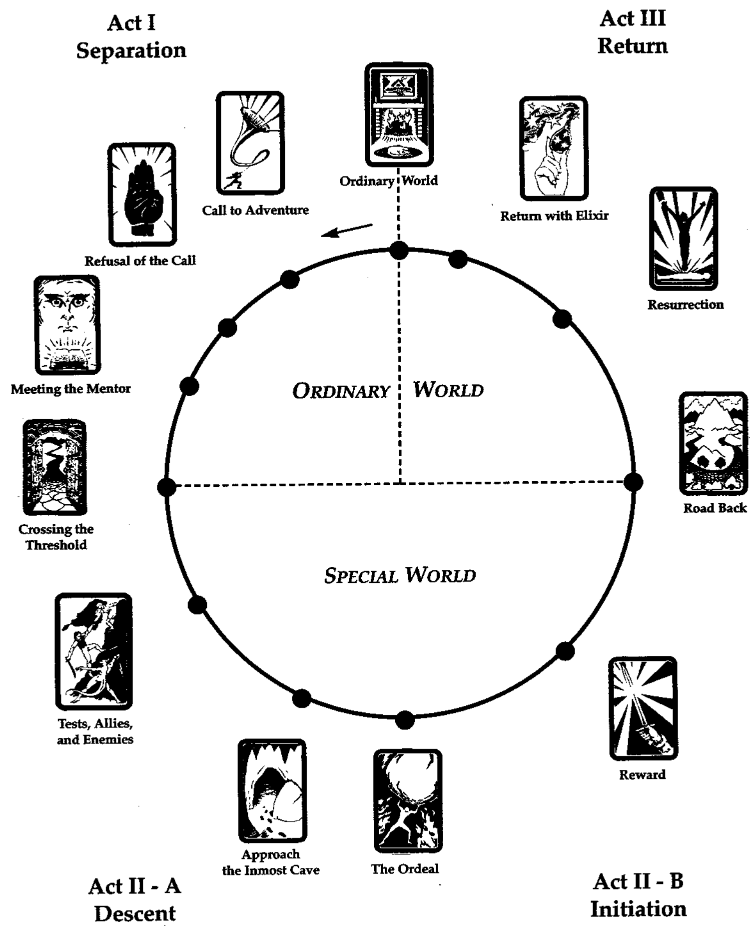


Figure 5. Vogler's Hero's Journey Model from [Voytilla, 1999]

D. INTERACTIVE ENTERTAINMENT

Interactive entertainment is a very general notion and determining what constitutes interactive entertainment is subjective. For instance, all of the following can probably be considered a form of interactive entertainment: participating in an academic debate, coming to bat in a baseball game, and riding a magic carpet in Disney's virtual theme ride Aladdin [Pausch *et al.*, 1996]. For the purposes of this dissertation, the discussion is limited to interactive entertainment in the computer-based domain. But even this encompasses a broad range of applications, including early adventure games, hypertext stories, video games, multi-user dungeons (MUDs) [Weyhrauch, 1997] and networked virtual environments, just to name a few.

Early forms of computer-based interactive entertainment took the form of text-based adventure games. In games such as Zork [Lebling *et al.*, 1979], the user acts as an explorer and solves puzzles to gain additional information and achieve some end goal. The user would enter commands at the keyboard and the system responded with text output describing what the user could see, hear and feel. This game presented an immersive text-based setting with its own consistent reality, but there was no real plot. Most of the stimulation came from solving the puzzles. The user was provided with a sense of increasing anticipation as the level of difficulty increased. But in the end, these systems did not tell a story, nor was telling a story of primary concern to the developers. The experience was interactive in that the user's choices affected the events of the game, but the user did not really interact with the characters. The source of interaction was the challenge of navigating through an unseen maze, in search of a single exit point. Paths through the game either led to dead ends or to a single prized ending. These programs were engaging in the mental challenge they presented their audience, but they provided little in the way of plot.

Hypertext stories are somewhat similar to early adventure games in that the reader is presented a maze they must navigate. Though in the case of hypertext stories, it is a maze of story pieces. Hypertext trades some of the interactivity offered by the adventure games for a much stronger story structure. The reader follows a path through the network of story pieces assembled by the author. The interaction occurs as a result of the user selecting from a small number of choices at distinct branch points provided by the author via hyper-links. Between each of the hyper-link branch points, the reading of the text is a linear experience. While the user's choices dictate the outcome of the story, it is the author who controls the flow. The network of possible stories is established at compile time and is static, resulting in a small finite number of paths or stories. Compare this to a collection of books where each book is based on a unique path through the network of the story pieces. [Weyhrauch, 1997] describes a path through a hypertext story as analogous to reading a single book selected from the collection.

Hypertext provides a solid story experience, but falls short of adventure games in its level of interactivity. By providing the reader a small number of choices intermixed

between long periods of linear text, the reader rarely feels as though they are participating in the story. The goal of interactive story is to immerse the user in the story world and make them feel as though they are an integral part of the action.

Video games are an example of interactive entertainment that are high on interactivity and sensory input but low on story. Playing these games can be a very powerful experience particularly with the advances in computer graphics. These systems appear more and more lifelike with the introduction of each new generation of hardware, providing a powerful sense of immersion. Soon, a computer-generated scene will be comparable to a movie scene, at least from a visual sense. While visuals are extremely important, they are only part of the experience. On the whole, video games lack meaningful character interaction and have little or no story structure. Even when there is an underlying story, it is merely to set the stage for the action and has little use throughout the remainder of the game. Video games rely on visual, aural and haptic stimulation, along with highly interactive action to draw the user into the game. An interactive story system strives to capture the immersive qualities of video games while providing meaning to the action.

Finally, the interactive worlds of Multi-User Dungeons (MUDs) allow distant players on the Internet to share a common virtual space in which they can chat with one another. Players improvise scenes together and imagine fictional worlds. MUDs are environments for fantasy play that allow people to create and sustain elaborate fictional personas over long periods of time. They are truly interactive in that they are meant to be experienced in the first person, not to be viewed or watched by an outside audience. However, MUDs fail as an interactive story because there is no consistent story behind the interactions. It is much like improvisational acting where characters (users in role as their MUD persona) explore the environment, interacting with one another in a manner consistent with their fantasy role.

While interactive entertainment can be highly immersive, it tends to regard interactivity and story as separate and almost incompatible. Interactive drama strives to meld the two, whereby the story being presented is inextricably linked to the actions of the user.

E. INTERACTIVE STORIES

Previous efforts at developing an interactive story system have met with limited success. Brenda Laurel [Laurel, 1986] proposed a design for an interactive fantasy system, identifying the system components and necessary functionality, but a system was never developed. More recent work as part of Carnegie Mellon University's OZ Project introduced the notion of a plot graph, a partial ordering of plot events [Kelso *et al.*, 1993] [Weyhrauch, 1997]. Weyhrauch used the plot graph as a foundation for searching all possible orderings of plot events in order to construct a story. While this approach successfully showed that it was possible to build an interactive story, it did not scale well to more complicated stories. The University of Southern California Institute for Creative Technologies (ICT) has developed a network-based approach to producing interactive training scenarios [Swartout *et al.*, 2001]. ICT's Mission Rehearsal Exercise (MRE) system allows users to experience a story in a highly immersive environment. However, user interaction points are fixed and tightly constrained in order to ensure the story conforms closely to the training objectives of the scenario. A more detailed description of these works is presented later in this chapter.

Other approaches to computer-generated stories have explored the use of story pieces in the form of standalone audio, video or text clips, to tell a story. Kevin Brooks of the Interactive Cinema Group at MIT's Media Lab uses a single set of predefined story pieces to tell many different linear stories [Brooks, 1999]. He uses story agents to arrange the pieces, resulting in stories that take on the personality of the respective story agent.

Developing a software program to tell a pre-scripted linear story might present some artistic challenges, but in the end it would be a straightforward effort. This is because the software engine driving the story has complete knowledge of the script and can exercise total control over the environment and the characters. When interactivity is introduced into the equation, the software no longer has control over the story. One of the characters (the user) is now free to act based on what has previously occurred, their interpretation of these previous events, and their perceptions of where the story might be headed. In spite of this, the system hasn't necessarily forsaken all control. It can limit what the user sees and constrain the user's options so as to nudge the action in a

particular direction. So the introduction of interactivity requires a tradeoff in narrative control. At one extreme there is unbounded interactivity. The danger here is that the user can be presented with so many options that they flounder aimlessly, never moving forward with a coherent story. Alternatively, if the user is provided no flexibility, the experience is no different than watching a movie or reading a book. In designing an architecture to control the flow of action in an interactive story, a balance must be struck between interactivity and narrative control.

In his dissertation, Peter Weyhrauch describes four approaches to guiding the action in an interactive story system [Weyhrauch, 1997]. The first is to build an environment with a great deal of interactivity and action, but no drama component at all. The idea is that if there are enough things going on, then something interesting is bound to happen. MUDs tend to fall into this category. A second approach is the use of an implicit dramatic structure. The structure follows from the characters and their goals. Video games and military simulations are examples of this approach. In a military battle simulation, the two sides have goals of conquering each other; this provides an implicit structure to the action. In these first two approaches, there is no real attempt to impart a narrative structure onto the action and interactions. If a story emerges, it is a by-product of the action.

The third approach Weyhrauch describes is a fixed story sequence. The user has some level of freedom, but there is a central director that ensures fixed story events occur. With this approach, narrative structure takes precedence over interactivity. Tinsley Galyean describes this as “narrative guiding interactivity” [Galyean, 1995]. The fourth approach is to use an explicit drama manager to guide the story events. The drama manager provides input to the characters and adjusts the story environment to encourage the action to follow a narrative structure.

These last two approaches to guiding action lead to two general categories of work in interactive narrative. The first is aimed at creating and presenting a specific story. That is, the user experiences the story the author has written. While it is possible and desirable for the plot to adjust and react to the variations resulting from the user’s interaction, in the end, the author’s story is told. The variations are constrained to ensure

that critical plot points are met. The second category of work focuses on dynamically building a plot based on the user's interaction. In some sense, the user builds their own story. The events and actions that occur are constrained both by the realm of possibility of the story world and an overriding, high-level narrative structure. It is the function of the drama manager to impart the narrative structure on the events of the story world.

1. Centralized Drama Manager

The first formal description of an interactive story system based upon a centralized drama manager was presented by Brenda Laurel [Laurel, 1986]. Laurel proposed an "interactive fantasy" system whose primary components were comprised of a centralized drama manager, a world model (or story world), system characters with a rich set of personality traits, and an intelligent user interface into the story world. At the heart of the system was a rule-based drama manager called PLAYWRIGHT. It was PLAYWRIGHT's job to give each character their formal specifications for the next incident. In turn, the characters would provide their suggestions for action back to PLAYWRIGHT that would implement the first acceptable suggestion. While Laurel's system was a theoretical design and never implemented, it was the first thorough treatment of the issues involved with developing an interactive story system.

A research group at Carnegie Mellon University has an ongoing project to develop an interactive drama system that seamlessly combines believable agents with an interactive story structure [Bates, 1992], [Kelso *et al.*, 1993], [Mateas, 1997] and [Mateas and Stern, 2000]. This group, known as the Oz group, works with the overriding philosophy that drama is a combination of story, characters and presentation. They focus on building worlds that give equal attention to believable agents, interactive plot and presentation. Fundamental to their notion of an interactive plot is the use of a central drama manager. Figure 6 shows the high-level architecture of the Oz project taken from [Mateas, 1997]. The *world* contains characters that exhibit rich personalities, emotion, social behavior, motivations and goals. The user interacts with this world through a *presentation* medium. Finally, the *drama manager* guides the experience of the user and makes things happen.

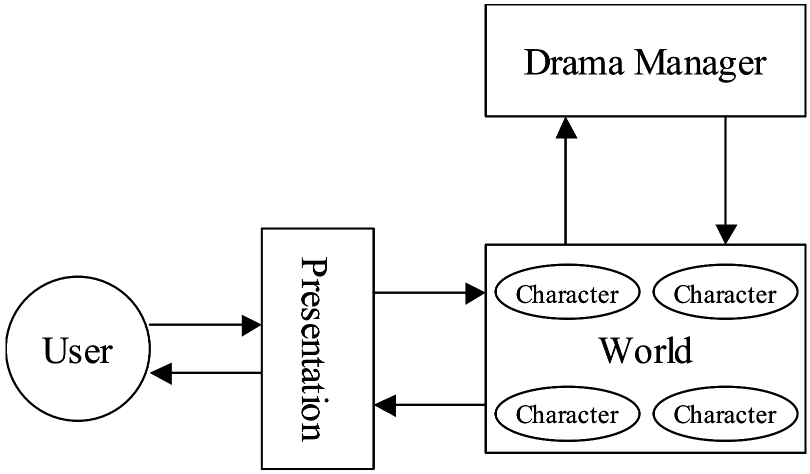


Figure 6. Oz Project Architecture from [Mateas, 1997]

Central to the idea of interactive stories is the idea of believable characters. This section will touch upon the work with believable agents that populate the *world*, but will focus primarily on the *drama manager*. A more thorough treatment of believable agents can be found in [Loyall, 1997]. Likewise, much of the Oz project work on *presentation* deals with generating English narrative [Bates and Kantrowitz, 1992] and is not discussed here.

a. Believable Agents

First, the concept of "believable character" must be established. A believable character is one who seems lifelike, whose actions make sense and who allows the user to suspend disbelief. Note that nowhere in the description is anything said about realism. For the purposes of drama and interactive stories, the character is believable if he or she remains in role and exhibits consistency of action, motivation and thought. Consistency is key to making a character believable. As described in [Mateas, 1997]:

> A character may be smart or dumb, well adapted to its environment or poorly adapted. But regardless of how "smart" a character is at dealing with their environment, everything they do, they do in their own personal style.

Once a character is developed in a story, the user forms certain expectations as to how that character will behave and react. The character must follow these expectations to remain believable in the eyes of the user. An "out of character"

response or unwarranted action serves to break the users suspension of disbelief. Believable characters are just that, believable, they are not necessarily realistic.

Bryan Loyall of the Oz group defined the following set of requirements for believable agents [Loyall, 1997]:

- Personality – Personality describes the unique and specific things about the character and is what makes the character interesting.

- Emotion – Characters must appear to have emotional reactions and have a means of expressing the emotions.

- Self Motivation – The agents must not only react to external stimuli, but also engage in actions on their own accord. They must be able to act based on internal drives and desires.

- Change – Characters must grow and change over time consistent with their personality.

- Social relationships – Characters interact with one another and these interactions are influenced by whatever relationships the characters have with one another.

- Consistency of Expression – To be believable, every avenue of expression available to the agent must convey a unified message consistent with the agent's personality and emotion.

- Illusion of life – This is a collection of requirements that includes pursuing multiple goals, capable of parallel action (e.g., walk and talk at the same time), possessing capabilities such as movement and perception, and finally the ability to react to stimuli in the environment.

These requirements led to the development of an agent architecture for believable agents called Hap [Loyall, 1997], [Bates, Loyall and Reilly, 1992], [Loyall and Bates, 1991]. Hap is both a language for defining agents and a software engine for controlling the agents. The language allows the author to define goals, behaviors and actions. At the heart of the architecture is an *active behavior tree* (ABT). The ABT is the main processing data structure in a Hap agent. Hap executes by repeatedly performing a multi-step loop that processes the ABT, expanding or shrinking the tree based on actions that have been completed, changes in the world and goals that have been

suspended. Once the tree is updated, Hap chooses the next action for the agent from the set of available actions in the ABT. The selection is based upon an internalized priority scheme with preference given to working on the same goals and behaviors that the agent had been recently pursuing. This priority scheme serves to maintain a sense of focused purpose to the agent's activity.

The follow-on to Hap, ABL (A Behavioral Language), is being used as the behavioral model for the believable agents in an interactive drama project entitled Façade. ABL closely follows the Hap semantics. A detailed description of ABL and the differences ABL and Hap can be found in [Mateas and Stern, 2002].

b. Drama Manager

The purpose of the Oz drama manager is to sequence major story events (plot points) into a coherent story line, while taking into account the actions of the interactive user, thereby achieving an interactive plot. The Moe architecture for dramatic guidance was developed for this purpose [Weyhrauch, 1997]. Moe functions by controlling the sequencing of major events (plot points) throughout the story. Moe uses an adversarial search technique, not unlike those used by chess games, to generate sequences of plot points. It then applies an evaluation function to rate each sequence. The highest rated sequence is used to select the next event in the story.

A plot sequence is made up of Moe moves and user moves. User moves are the actions that the interactive user can take in the story. For example, in his dissertation [Weyhrauch, 1997], Weyhrauch uses a mystery story to demonstrate the Moe architecture. Examples of possible user moves would be to search the murder scene, discover a piece of evidence or confront a suspect. The set of all possible user moves are formed into a plot graph. A plot graph is a partial ordering of story events linked together in a directed acyclic graph. The plot graph provides a basic hierarchical structure to the events. For instance, in the plot graph, the event of sending evidence to the crime lab for analysis comes after the event of finding the evidence. The plot graph is used for generating a set of legal user moves from which to select the next move. The actual selection occurs though the adversarial search according to an evaluation function.

Moe moves are those actions the drama manager can take to guide the story and move it in a desired direction. An example of a Moe move might be to create a

commotion so as to attract the user's attention so they stumble across a piece of evidence. The intertwining of the Moe moves with the user moves results in a completed plot sequence or scenario.

Weyhrauch developed an evaluation function to rate scenarios based on a weighted linear combination of the following seven factors:

- Thought Flow – A measure of whether one user event relates logically to the next event.

- Activity Flow – A measure of how bored the user might feel by moving around with nothing occurring.

- Options – A measure of how much freedom of choice the user perceives they have.

- Motivation – Measures whether the user's actions are related to their active goals.

- Momentum – Certain events make sense if they occur in close proximity to one another. Momentum provides a measure for this. The story's author determines which events are related and how close in time they should occur.

- Intensity – Measures the building of excitement as the story progresses. This captures the classical dramatic arc discussed earlier in the chapter.

- Manipulation – A measure of how manipulated the user feels by the drama manager.

It is important to note that the evaluation function only works on complete scenarios. Logically, a scenario is made up of two parts; the first consists of the fixed sequence of events that have already occurred and the second is some sequencing of the events that have yet to occur. Beginning with the sequence of events that have already occurred, an adversarial search technique is used to explore all possible sequences of events yet to occur (Moe and user moves). The search tree is expanded by considering all possible Moe moves and legal user moves (the plot graph is used to determine which user moves are legal). The expansion continues until every user move has been used. Each of these completed scenarios is evaluated, the best one selected, and this determines

the next move in the scenario. The process then starts all over again. This type of algorithm results in a tree with a tremendously high branching factor. Even with a small number of user moves and Moe moves, the problem of searching the entire tree is intractable. For example, with 15 user moves and 15 Moe moves, there can be as many as 30! (2.65×10^{32}) scenarios that must be initially evaluated. As the story progresses and the history of events becomes fixed, the complexity of the problem decreases, but not fast enough to make it manageable. As a result, Weyhrauch developed two heuristics for rating scenarios in real time. The details of the heuristics are not described here but can be found in [Weyhrauch, 1997].

Weyhrauch's work showed that an interesting and dramatically appealing story can be presented while remaining true to the user's interactive freedom. However, the plot graph paradigm and use of an adversarial search with user and Moe moves does not scale well. The architecture is quickly overcome by stories involving any sizeable number of plot events. Additionally, his work required that every possible event in the story be defined in advance and that every one of these events be used in the final scenario.

Using the Moe system, every story that occurs will consist of the same basic events, the only difference being how the events come about and in what order. The ending is always the same, only the path to that ending is variable. It is this variability that is at the heart of the interactive nature of his architecture, but in the end, all paths must lead to the capture of the murderer.

The multi-agent system based story world approach described in the previous chapter is intended to present an *interesting* story, given the characters and constraints of the story world, not a *specific* story. In the case of a mystery, there are many good stories that can be told, not all of which end with the perpetrator being caught. For example in the 1999 film *Entrapment* with Sean Connery and Catherine Zeta-Jones, Connery plays a resourceful master thief and Zeta-Jones plays an insurance investigator who sets an elaborate trap to capture him [Ebert, 1999]. A traditional script would result in Connery being caught and sent to jail. However, in the end Connery escapes capture by a strange twist of romantic intervention on the part of Zeta-Jones. Within the story

world of *Entrapment*, it is not difficult to imagine other exciting stories with different conclusions.

2. Dramatic Beats

Recent work by Michael Mateas of the Oz group and Andrew Stern has moved away from the plot graph paradigm for sequencing story events [Mateas and Stern, 2000], [Mateas and Stern, 2002]. They propose the design of an architecture for integrating plot and characters that is based on a dramatic beat. A beat is the smallest piece of dramatic action that can occur. Scenes are composed of multiple beats. The execution of a beat causes some "value" within the story environment to change, where a value is a property of an individual or relationship. Associated with each beat is a set of preconditions necessary for the beat to occur, a description of the values to be changed by the beat, conditions used to determine success or failure of the beat, and joint plans to be carried out by the characters in order to execute the beat. The joint plans are simply the behaviors necessary for the characters to carry out the beat.

When the preconditions have been met for a specific beat to occur and that beat is chosen for execution, the drama manager accesses the "joint plans" for the specific beat and hands the appropriate characters their behaviors necessary to carry out the beat. The plans have been designed to accomplish the beat. This means that high-level goals and plans that drive an agent's behavior do not reside within the agent, but are located in the beat. The drama manager parcels out the behaviors and goals to the agents participating in the beat. Low-level goals still reside within the agent (movement, personality moves, facial expressions, etc.). A set of related beats necessary to complete some larger action are grouped together in a scene.

A scene consists of preconditions, a description of the values intended to be changed by the scene, a collection of beats with which to construct the scene, and a description of the dramatic arc that should be followed within the scene. Preconditions test whether the scene is appropriate given the current story and character state.

Sequences of scenes are chosen by a drama manager. At any given time, the story is in a certain state consisting of the current story values and other global state information such as active conversational topics, physical locations of characters, etc.

The drama manager is continually aware of the story state and chooses the next scene by examining the list of unused scenes and selecting the one that satisfies the preconditions and whose value changes best match the global plot arc.

3. Directed Improvisation

Barbara Hayes-Roth describes improvisation as “a particular form of theater in which actors spontaneously and cooperatively generate their stories and their characters at performance time under the constraints of directions from sources such as the audience, predefined scenarios and other actors” [Hayes-Roth and Rousseau, 1997]. Stanford University’s Virtual Theater Project (VTP), under the direction of Barbara Hayes-Roth, has developed an architecture for building computer characters that perform directed improvisation. Synthetic agents are provided as intelligent actors that improvise their behaviors without detailed planning. The underlying agent architecture centers around a *mind-body* design. The *mind* is the implementation of a social-psychological model integrating personality traits, moods and attitudes affecting interpersonal relationships. The *mind* updates the agents’ knowledge based upon external inputs and stimuli, controls the agents decisions and provides input to the *body*. The *body* is the expression of the actions selected by the *mind* [Hayes-Roth and Rousseau, 1997], [Hayes-Roth *et al.*, 1995].

There are two types of improvisation characters: autonomous actors and avatars. The avatar provides the user with an interface into the story world. Both the actors and the avatar receive directions from a scenario and other actors. The autonomous actors decide their behavior based solely on their personality, mood, attitude, and the received directions. The avatar is primarily directed by the user who selects the actions to perform. However, the manner in which the avatar carries out the selected action is determined by the avatar’s personality, mood and attitude.

A predefined story scenario is used to impose a story structure on the action. In an application of the Virtual Theater called CyberCafé, a restaurant scenario is applied with two autonomous actors and an avatar. One autonomous actor is a waiter, the other is a customer. The avatar plays another customer. The scenario provides the high-level direction to the actors, so as to cause major events to occur in a prescribed order. The

specific actions chosen by the autonomous actors to carry out the events are determined by the actor's personality, mood, and attitude. Depending on the mood of the customer when the waiter brings his drink, the customer may do anything from throwing the drink back at the waiter, to thanking him and providing a tip. The variability in the selected actions manifests itself as improvisation.

The work of the VTP deals with a specific type of drama, focusing heavily on character and less on variability in the plot. In fact, the plot is defined in advance and remains static throughout the story.

4. Verb-Centric Interactive Story-Telling

In the Virtual Theater Project, Barbara Hayes-Roth capitalizes on the unique personality and motivations encoded into each agent to tell a story from a fixed script. The characters' mannerisms and personalities are driven by their interactions with each other and the user's avatar. The stories are interactive because the agents' moods and reactions are affected by the environment and the other characters. The character interactions take on a story form through the imposed script.

While still relying heavily on believable agents that react to their environment, the Oz project takes story plot a step further, allowing the plot to change based on previous events and the prognosis for the remaining events forming a good story. Chris Crawford has designed a storytelling system, Erasmatron that takes a wholly unique approach to interactive storytelling [Crawford, 1999].

Crawford's approach is founded in the assumption that storytelling is inextricably the function of an artist, for which no algorithm can be designed to truly replace the human story teller. Storytelling relies heavily on contextual knowledge of the culture in which it is told. Crawford states "Just as the meaning of language is steeped in culture, so too must the story teller integrate the story in the audience's culture. For this reason, interactive storytelling with all of its creative responsibility must always be the domain of the artist and not an algorithm" [Crawford, 1999].

Erasmatron, as described by Crawford, is a verb and sentence based system. The story is organized around a master list of verbs for the story world. For each verb, the author declares a set of character roles necessary to carry out the verb action. For

instance, a sentence with an action verb clause of "assault" might have character roles of attacker, victim and bystander. In this system, artistic functions are segregated from algorithmic functions. The artist sets dramatic goals (artistic function) that require the characters to take action, while an algorithm handles the physical implementation or realization of the action. Continuing with the verb clause "assault," for the attacker to accomplish their goal they will need to move to the vicinity of the victim. The artist initiated the character's need to move, and an algorithm controls the character's movement.

Dramatic goals are established through inclination functions coded by the author. These functions form the basis of the character's decision process. When faced with a choice, the autonomous characters reference an inclination function to resolve the decision. Erasmatron provides the environment for designing the story world, but the definition and coding of the inclination functions is left to the artist.

While the system allows for complex behavior and artistic influence, Crawford acknowledges "I have been hoisted by my own petard in the matter of complexity; as yet, largely because of its complexity, not a single artist has completed creation of a viable story world using the Erasmatron!" [Crawford, 1999]. As a result, he has moved to an intermediate stage he calls "assisted storytelling." The latest version of Erasmatron has stepped back from requiring the artist to code inclination algorithms to govern the choices for the characters. The artist's role is now reduced to defining dramatically viable options for the characters, while the user makes the actual choices for the characters.

5. Interactive Stories with Strong Pedagogical Goals

As previously discussed, interactivity and plot can be balanced in a number of ways. In Hayes-Roth's VTP, interactivity is expressed through the diversity of agent responses to a given situation. Interactivity occurs at the action selection level, while the basic plot remains unchanged. In the Oz project, interactivity manifests itself at the plot level. By defining a set of plot events and computer events that will tell the story, interactive input by the user impacts the sequencing of plot events in real time. The script changes, but only in the ordering of events. A dramatically appealing story

emerges as a result of the drama manager's selection process of events. In this section, two approaches are examined for presenting stories with strong pedagogical goals. In both cases, achieving the prescribed learning objectives is of prime importance, as a result the interactive freedom of the participant is constrained to maintain the integrity of the script.

a. A Story Deconstruction Approach to Interactive Drama

A group of researchers from the Center for Advanced Research in Technology for Education (CARTE) at the University of Southern California's Information Sciences Institute have been working on an agent-based approach to creating interactive pedagogical drama [Marsella *et al.*, 2000]. In their system, interactivity is provided on two levels, the character level and the story event level, while the narrative structure of the story is maintained through a script deconstruction scheme. Story characters are free to choose their actions autonomously, while director and cinematographer agents manage the action and its presentation in order to maintain story structure, achieve pedagogical goals and achieve the best dramatic effect. There are four main components to the system: autonomous character agents, the user or person learning, a director agent and a cinematographer agent. These components have been brought together in a multimedia title called *Carmen's Bright IDEAS*.

The story is provided in a "presentational" style where the user controls the intentions of one of the characters, but does not participate directly in the first person as a character. This allows a level of interactivity specific to the user's characters.

Event level interactivity is introduced by starting with a full story script and successively decomposing it into smaller and smaller pieces, all the while identifying places where variability can be introduced while remaining true to the pedagogical goals of the script. This allows interactivity in the plot while maintaining the story structure. Fundamental to the decomposition process is determining which variations are desirable, either from the pedagogical perspective or from the dramatic perspective. This helps in defining the actions of the director agent to avoid undesirable variations.

The script is a sequence of acts, where each act is a sequence of scenes. For a scene to be realized, certain events must occur within the scene. The events themselves are motivated by the goals of the individual characters participating in the

scene. The script is broken down into a hierarchical narrative structure, starting with the acts and continuing all the way down to analyzing the goals that can cause the events to occur that make up a scene. Once decomposed, the designers determine where variability can be introduced. For example, alternative scenes or a different ordering of scenes may be able to achieve the pedagogical goal of the act. Similarly, different patterns of events can achieve the same scene goal.

It is the job of the director agent to select the specific actions for the agents to present a dramatically appealing story. The director in essence controls a discrete event driven simulation, where the discrete events are individual character dialog turns. The director has global knowledge and selects the next event based upon the internal state of each character and state of the environment. With this state information, the director selects the actions for the characters based upon the previous script decomposition.

A cinematographer agent manages the presentation of events to maximize their dramatic impact. The agent gets a filming description from the character agents at each dialog turn. The filming description sets the action that needs to take place in the next turn. The description contains information such as who is speaking, what is said (and how long it will take), non-verbal gestures that will occur, any dialog boxes (characters thought boxes) and flashbacks that must be shown. Based on this, the agent references a set of predefined rules to determine what to show, how to show it and when.

By starting with a well-defined script, the story remains true to the pedagogical goals while supporting interactivity at the plot level. In essence, the decomposition process yields different means to the same end.

b. Predefined Networks with Confined Freeplay

The Institute for Creative Technologies at the University of Southern California is a U.S. Army funded institute with a mandate to bring together the resources and talents of the entertainment and game development industries with computer scientists in order to advance the state of immersive training simulation [ICT, 2002]. A keynote project of ICT is the Mission Rehearsal Exercise (MRE) system, a virtual reality training environment that combines virtual humans [Rickel *et al*., 2001], wide screen graphics and immersive audio to present a real-world training scenario. Underlying the

MRE system is a network-based interactive story system called StoryNet [Swartout *et al.*, 2001].

As previously discussed, interactive narrative carries the inherent problem of balancing narrative control with interactivity. In the case of military training systems such as the MRE system, this challenge is exacerbated by the need to ensure doctrinally correct training objectives are met. These training requirements place an additional restriction on the level of interaction available to the participant. ICT's approach is similar to that used in CARTE's *Carmen's Bright IDEAS*. The MRE system starts with a well-structured script that meets the prescribed pedagogical goals. The script is then structured into a network of interactive "freeplay" nodes and linear transition links that are used to setup the follow-on "freeplay" node. Associated with each transition link are gating conditions used to determine when the participant has met the conditions necessary to transition to another node (Figure 7).

In this system, the nodes correspond to tasks the participant in intended to learn, or key decision points in the scenario. Within the nodes, the participant is free to interact with the virtual humans and make decisions. Extending from each node are linear movie links. Once the interaction is complete and the participant has met the gating conditions (either through a specific decision or completion of a task), they take on a passive role while being transported to the next node via one of the linear movie links. By limiting the options available in the freeplay nodes and the number of links extending from each node, the designers are able to control the scope of behaviors the system must handle.

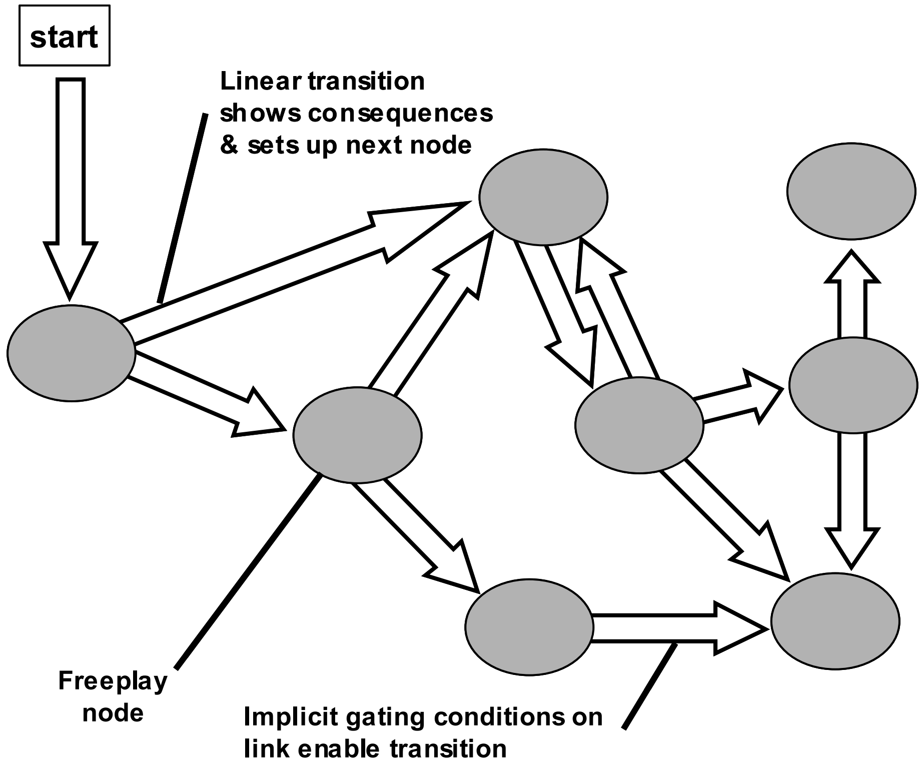


Figure 7. StoryNet Architecture from [Swartout *et al.,* 2001]

The MRE system is an extremely effective training environment. The 8.75 by 31 ft. curved screen, coupled with intelligent characters, real-time animation and 64-track spatial sound creates an almost holodeck-like environment. However, this level of immersion comes at a price in terms of story complexity and plot variation. Given the extensive scripting required to meet the training goals and develop movie links, expanding the system appears to be both time consuming and expensive. Adding a new decision option to an existing node requires a script to be written and a linear movie developed for the associated outgoing link. Similarly, if an entirely new node is added to the network (new decision or training objective), a host of links and supporting nodes must be added.

F. AGENT-BASED PLANNING SYSTEMS

Generally, multi-agent systems can be divided into two kinds of planning schemes: static and dynamic. Planning is considered dynamic if the plan evolves during execution of the simulation model. Static plans are pre-arranged, i.e., defined before simulation runs take place.

A hybrid approach combines static and dynamic planning as proposed by a number of others, most recently in MEAGENT [Rowe *et al.*, 2000], [Andrade, 2000]. MEAGENT is a Prolog system using means-ends analysis. MEAGENT agents are given tree-structured plans by their designers, but then allowed to adapt. Agents are given plans that are designed to seek goals by dividing the goal into sub-goals, and sub-goals into smaller sub-goals, etc. The root of the goal tree represents achievement of the top goal, and the leaves of each tree represent one step in achieving a sub-goal. When appropriate, the tree is re-structured, dynamically, in a process that Rowe and Andrade call "re-planning." Re-planning in MEAGENT was used in training sailors how to fight fires onboard ships.

MEAGENT adapts to unexpected events during the simulation, but it does not invent plans. Each simulation run begins with a static pre-defined plan (goal tree), which must be designed by the human operator. While means-ends analysis provides flexibility and adaptability, it doesn't solve the problem of plan discovery.

Another approach that is closely related to this work is PLANGENT [Oshuga *et al.*, 1997]. Mobile agents in PLANGENT have planning capabilities that are a by-product of adapting to new web-based environments. The authors claim that this kind of adaptability and re-planning make agents "intelligent."

According to [Oshuga *et al.*, 1997],

> PLANGENT has an advanced planning feature – specifically, a technique that uses backward reasoning from declarative statements of user requirements to generate sequences of actions that will satisfy the requirements. The planning mechanism is reflective, that is, agents can execute metalevel planning, or meta-planning.

PLANGENT allows incomplete plans, thus, actions can be delayed until additional information is gathered by the mobile agent, and the action resolved during execution. PLANGENT agents can make a *least commitment* plan that delays definition of which action to perform until the agent has visited enough web sites to decide. Still, plans are limited by top-level requirements as defined by the user. PLANGENT's dynamic planning steps are described in Figure 8.

```
1. Express user requirements in terms of constants,
   variables, and predicates.
2. Initialize a plan as a set of actions and constraints.
3. The agent tries to satisfy an un-realized goal by
   choosing an action from a plan or by choosing an action
   that is not in its original plan.
4. If an action was selected in step 3 that is not in an
   original plan, that action is considered a threat if it
   breaks the consistency of the original plan. A threat
   action is handled by gathering more information or
   backtracking. If no more actions are available, the
   agent backtracks to step 3.
5. The current plan is executed.
6. Generate a metaplan: If pre or post conditions are not
   met, declare failure. If infinite loops are found,
   terminate the planning process. If failure, re-planning
   is required.
```

Figure 8. PLANGENT Planning Steps from [Oshuga *et al*., 1997]

PLANGENT has been applied to airline reservation systems where goals are destinations, travel dates and times, and constraints are cost, etc. Plans are represented as trees with sub-goals derived from top-level user input requirements.

G. SUMMARY

In this chapter, work related to the design and implementation of interactive story systems was presented. An overview of interactive entertainment and narrative intelligence was provided along with a description of plan-based story-telling and story-understanding systems. While this work served as a forerunner to interactive stories, many other fields of study have played key roles or provided motivation behind research into developing interactive story systems. Some of these include human-computer interface design [Laurel, 1991], knowledge understanding and representation [Schank and Abelson, 1977], AI planning [Young, Pollack, & Moore, 1994], narrative theory [Aristotle, 330 BC] [Freytag, 1900] [Polti, 1977] and interactive cinema [Brooks, 1999] [Davenport *et al*., 2000]. The range of influence is too great to present a thorough treatment of each of these in this dissertation.

The interactive story systems presented in this section were described with an emphasis on the approach used to balance interactivity with narrative control to present a story. This examination reveals that existing systems either don't scale to large complicated stories (i.e., only a small number of stories are possible), or they are based on a fixed plot. A scalable architecture supporting variable story lines that can be

adapted to multiple simulation domains does not yet exist. The goal of this research is to develop a story world where literally thousand of stories are possible. To accomplish this, the top-down structure employed by previous systems has been abandoned in favor of a bottom-up design paradigm based on multi-agent system simulation techniques.

While a review of the previous work revealed areas requiring attention, it also served to provide inspiration, both in terms of what has worked in the past, and in what has not. The common theme of a central drama manager shared by virtually every project described, and the inherent problems with scalability, led to the exploration of a bottom-up, distributed control architecture for generating stories. A second common thread running through a majority of the projects was the notion of a fantasy world, or a coherent space in which the story (or scene) takes place [Laurel, 1986], [Weyhrauch, 1997], [Mateas, 1997] and [Hayes-Roth and Rousseau, 1997]. The concept of a story world evolved into an environment for a multi-agent system. Finally, a study of related work, as well as additional readings in narrative theory and interactive narrative [Murray, 1998], [Polti, 1977] revealed an obvious, but fundamental relationship between an actor, character, role and scene. An actor takes on the personality of a character, while the actions available to the character are a function of, and bounded by, the scene and the role the character plays in the scene. This actor, character, role, scene relationship, as it applies to the story engine, is described in Chapter VI.

The remaining chapters present a scalable multi-agent system architecture that balances interactivity, with character autonomy and story world controls, to present the user with a personalized story.

III. MULTI-AGENT SYSTEMS

A. INTRODUCTION

This chapter presents a definition of multi-agent systems (MAS) that serves as a basis for defining the MAS architecture used by the story engine to generate interactive stories.

MAS are composed of numerous interacting computing elements, known as agents. Agents are computer systems with two important capabilities. First, they are, at least to some extent, capable of autonomous action – of deciding for themselves what they need to do in order to satisfy their design objectives. Second, they are capable of interacting with other agents – not simply by exchanging data, but by engaging in analogues of human social activity: cooperation, coordination, negotiation, and the like [Wooldridge, 2002].

MAS operate from the bottom-up, using multiple adaptive agents "…(as) intelligent actors, interacting among themselves by using their defined attributes and methods, but (are) able to modify those constraints to meet the goals assigned them by the modeler…providing real insight into how best to encourage and take advantage of individual initiatives and adaptability " [Weiss, 1999].

MASs have no centralized control – the agent simulation is leaderless. Each agent in the simulation independently pursues its own independent goals. Some agents may cooperate, while others compete. The result is a highly dynamic environment where software agents, with no human intervention, can explore an environment, interacting with other agents and object in the environment in pursuit of their goals. The outcome is innovative solutions for achieving the goals [Hiles *et al*., 2001].

While there are many definitions of multi-agent systems with varying degrees of formality, Ferber's definition of a MAS (given in the next section) is used as a basis throughout this dissertation [Ferber, 1999]. Additional general information on multi-agent systems can be found in [Weiss, 1999], [Holland, 1995] and [Russell and Norvig, 1995]. Descriptions of specific MAS architectures and simulations can be found in Swarm [Langton, 1997], ISAAC combat simulation [Ilachinski, 1997], and Echo

simulated world [Echo, 2000]. These examples primarily explore the use of MAS architectures to model emergent behavior.

B. MULTI-AGENT SYSTEM DEFINITION

Ferber describes a multi-agent system as "an electronic or computing model made up of artificial entities which communicate with each other and act in an environment." More precisely, multi-agent systems can be defined as a set of interacting elements described by Equation 1.

$$MAS = \{E, O, A, R, Op, Laws\}$$

E – Environment
O – Objects situated in the environment
A – Agents, ($A \subseteq O$)
R – Relations linking objects O
Op – Operations
Laws – Constraints governing the environment

Equation 1. Multi-Agent System from [Ferber, 1999]

1. Environment

MAS simulations can be formulated in a situated or non-situated environment. In a situated simulation, the environment may be a Euclidean n-space or a notional space appropriate to the given simulation. Regardless, at any given time, the agents and objects have a "location" in the environment. This may be an (x, y, z) coordinate in the case of a 3-D environment, or it may be a more abstract "location" defined in terms of time and resources. In a situated simulation, agents are capable of perceiving their environment, recognizing the objects and agents populating the environment, and transforming the state of the environment by interacting with the other agents and objects.

In the case of non-situated MASs, the environment is populated with agents and objects that interact in accordance with defined *relations* through agent-to-agent and agent-to-object communications. The objects are typically resources needed by the agents to achieve their goals. The agents and objects need not have a "location." The MAS is defined by the behavior of the agents and the network of *relations* linking them together. Ferber describes this type of MAS as a "purely communicating MAS."

2. Objects

Objects are described as passive entities situated in the environment that can be perceived, created, destroyed and modified by the agents. Under Ferber's definition, agents are a subset of the objects with no clear distinction between the two. Objects can be thought of as non-agent entities in the environment. They may be totally passive like a rock or a tree, or they may be active as is the case with a radio transmitter emitting signals into the environment. The primary distinction between objects and agents is intent. Objects may be able to act to modify the environment, but there is no autonomous intent behind the actions.

3. Agents

Ferber describes an agent as a physical or virtual entity that can act in an environment, communicate with other agents, is driven by internal goals and objectives, possesses resources, perceives its environment but only a partial view (or possibly none at all), has skills, may reproduce, and behaves in a manner that satisfies its objectives. His definition makes no distinction between cognitive and reactive agents, and is general enough to encompass both.

Cognitive agents (or deliberative agents), from the distributed artificial intelligence (DAI) community, are traditionally based on first-order predicate logic, sophisticated reasoning, and rely on the internal manipulation of symbols. These agents maintain a symbolic representation of the environment within which they operate, and focus on communication and cooperation between agents. Most importantly, these agents have *intentions* – goals and plans to achieve goals [Hiles *et al*., 2001].

Reactive agents, from the field of artificial life (A-Life), are reflexive – actions are "reactions" to stimulus regulated by perceptions and the agent's internal state. These agents maintain no planning, history, or symbolic representation of the world. The simple reactive agents are combined into a society, where intelligence is seen as emergent from the vast interactions of the agents and the environment. See [Weiss, 1999] for a more detailed comparison of cognitive and reactive agents.

Throughout the remainder of this dissertation, both objects and agents are referenced extensively. Often it is convenient to make statements that apply to both

objects and agents. When both are being referred to together, the term *entity* will be used. From a software engineering perspective, an *object* is an *entity* and an *agent* is an *entity*.

4. Relations

Relations are abstract links that create a dependency between the agents. They can be as simple as allowing communications between two agents or they can be complicated relationships establishing the rules of cooperation and competition among a society of agents. For example, two agents can form a cooperative relationship to pool resources that will allow a common goal to be achieved that would not be possible by any one agent acting alone. In the case of a simulation involving a military command structure, the chain of command is a complicated relationship that imparts certain responsibilities on the agents based on their position in the structure. An agent filling a "command" role will inherit certain goals that go along with the command position. In addition, the command relation carries with it a certain amount of influence over the agents in subordinate positions; the goals of the "commander" agent influence the goals of the "subordinate" agents. A detailed discussion of relations as they apply to multi-agent systems can be found in [Roddy and Dickson, 2000].

5. Operations

Agents are capable of autonomous actions or "operations" that allow them to perceive their environment, interact with each other and objects, and change their state, either internally (mood swings from *happy* to *sad*) or relative to the environment (move from location x to location y). The set of actions available to an agent is dynamic. It is constantly changing based on the current context, where current context is a function of the agent's state and that of the environment. That which is appropriate at time t, may not be appropriate at time $t+1$. The change to the set of appropriate actions could be because the agent's state changed, or it may be the result of a change in the environment.

Given a MAS simulation with environment E, if Ω_a is the comprehensive set of possible actions for agent a, then Equation 2 describes the reduced set of possible actions a can take at time t. This set of actions, $\Omega_{a,t}$, is a function of agent a's state at time t, $\xi(a_t)$, and also the state of the environment that is within the given agent's scope of

knowledge $\xi(E'_t)$. E'_t is the agent's view or perception of the environment at time t. Included in $\xi(E'_t)$ is the state of all of the agents and objects in E' as perceived by a. As stated before, agents only have local perspective; they do not normally have complete knowledge of their environment so $E' \subseteq E$.

$$\Omega_{a,t} = f(\xi(a_t), \xi(E'_t)); \Omega_{a,t} \subseteq \Omega_a$$

$\Omega_{a,t}$ = *the set of valid actions for agent a at time t*
Ω_a = *set of all possible actions for agent a*
$\xi(a_t)$ = *the state of agent a at time t*
$\xi(E'_t)$ = *the state of environment E' at time t*

E' is agent a's perceived environment ($E' \subseteq E$)

Equation 2. Action Set for Agent Type a at Time t

While the set of all possible actions for an agent may be very large (Ω_a), the contextually appropriate actions at any given time ($\Omega_{a,t}$) should be a much smaller set. It is from this subset ($\Omega_{a,t} \subseteq \Omega_a$) that the agent decides what to do based on its active goals.

6. Laws

Given the operations described above, the rules for applying the operations, along with the reaction of the system to the operations, are captured in the laws of the system. These laws are the limitations and restrictions the agents and objects must adhere to while they reside in the environment. They might include issues related to physically based modeling such as collision detection, gravity and light propagation, or they may govern the way relations are created and destroyed [Roddy and Dickson, 2000].

C. SUMMARY

This chapter presented a general framework for describing multi-agent systems based on Ferber's definition. MASs were described in terms of their environment, the *entities* populating the environment (agents and objects), the relationships established between the *entities*, and finally the actions agents can take (operations) and the rules by which the actions are applied (laws). In the next chapter, the design philosophy underlying MAS research at the MOVES[2] Institute is presented along with specific design concepts that are key to this research.

[2] MOVES (Modeling of Virtual Environments and Simulation) Institute, Naval Postgraduate School, Monterey, CA.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. MULTI-AGENT SYSTEM RESEARCH

A. INTRODUCTION

For the past three years, the MOVES Institute's Computer-Generated Autonomy Group has been exploring MAS architectures that facilitate the development and control of complex, adaptive behavior. This chapter introduces four agent-based simulation design concepts, *composite agents*, *goal management*, *tickets* and *connectors*, for modeling multi-agent systems and implementing the models in software simulations [Hiles *et al*., 2001]. These concepts have evolved over the past three years into a novel simulation methodology called Connector-based Multi-Agent System (CMAS) simulation that is capable of generating dynamic plans and interactive stories. The CMAS architecture, described in Chapter V, serves as the underlying model for the story engine.

Portions of the following text have appeared previously in two collaborative papers: [Hiles *et al*., 2001] and [VanPutte *et al.*, 2001].

B. SEMI-FLUID SOFTWARE STRUCTURE

Interactive stories, along with their underlying plots, are ever changing and fluid structures that only exist in the past. That is, at any given instant in time, it is possible to precisely describe what has already occurred, but not will happen next. In essence, stories are generated one step at a time based on the complex interactions of the characters with each other and their surroundings. The structure of an interactive story is not fixed until after the story is told.

Software development has traditionally focused on building software that is in direct conflict with the notion of fluid structures. Software systems are typically engineered based on rigid designs where fixed and immutable relationships are established among the components inside the software. It is assumed that the structure must be tightly bound at design time if a program has any chance of meeting its design goals. This outlook can be described as analogous to a new highway system that is designed on paper and constructed with concrete and steel to meet the forecast needs of a growing city. Once built, the highway system remains fixed and static unless new

construction occurs. It would be absurd to expect it to mold itself into new forms to meet growing infrastructure and changing traffic patterns.

This same thinking underlies much of the work in developing plans and producing interactive stories. The architecture of the plot is fixed at design time; its structure is basically inert. While varying degrees of deviation are possible based on the systems' design, in general, the only stories that can be generated are those that the author or designer has thought of a priori. The systems are not capable of producing stories that were not laid out in advance.

With multi-agent systems, it is possible to build software that modifies its own structure, within a set of constraints, to maintain close contact with a dynamic environment. In the case of interactive stories, this entails a MAS architecture that allows the story system to generate a fluid plot that is sensitive to the dynamic relationships between the characters, and the characters with their environment, while remaining true to the basic constraints (or laws) of the story world.

1. Indirect Solutions - A Design Paradigm Shift

Most software developers and programmers have been trained in traditional software engineering, relying on structured system designs that implement a *direct* solution to the problem. Traditional problem solving in software engineering is *direct* in the sense that the developer conceives of an algorithmic solution and transfers that solution to software. Software development rigor and practice is used to ensure the code will produce an exact execution of the algorithm. In direct solutions, the programmer knows exactly how to solve the problem and the software implements that solution precisely. This approach is fine for problems where the domain is well known, and the relationships are static, finite and well defined. Direct solution systems are somewhat comparable to well-behaved functions; for a given input, the designer knows what to expect for the output. Surprises are a clear indication of a bug in the system.

In sharp contrast, surprises in MAS simulations are not only acceptable, but are the desired end, as long as the system operates within boundaries that are explicitly determined. The software is intended to surprise the designer within a system of constraints. This is possible through the use of software agents that discover an *indirect*

path to the solution, thereby allowing for the possibility of arriving at a solution the designer may not have previously considered. In this way, multi-agent systems are capable of producing innovative solutions. These solutions are *indirect* in that they were not explicitly programmed into the software; rather they are solutions that are consistent within the constraints the designer places on the software agents. As a result, any solution that is valid within the imposed constraints, is no longer a bug, but a potential novel approach to the problem. This is precisely the design philosophy that inspired the story engine. A story world is defined and novel story lines (*surprises*) are generated as a result of agents pursuing their goals (*indirect solutions*).

C. COMPOSITE AGENTS

Multi-agent system simulations typically consist of numerous high-level agents that represent entities operating in a common, shared environment. The agents residing in this "outer environment" interact with one another and the objects in the environment. They sense their environment, interpret the sensory input and decide what actions to take. These actions, in turn, affect the environment either directly through agent-to-environment interactions or indirectly through agent-to-agent interaction. In an effort to capture the strengths of both cognitive and reactive agents, while at the same time simplifying the design of such a complex agent, the *Composite Agent* (CA) architecture was developed.

CAs are composed of a combination of agents (Figure 9). They contain a set of *Symbolic Constructor Agents* (SCAs) that work with sensory streams (or *impressions*) from the outer environment to create a symbolic inner environment (E_{inner}) representing the agent's perspective of the outer environment (E_{outer}). The basic structure of a Composite Agent follows Wooldridge's "observe - update state - act" model for agents that maintain state [Wooldridge, 2002]. The SCAs define the agent's sensor capabilities and are tailored to sense specific aspects of the environment. They also act to control and filter *impressions* of the outer environment, so the agent isn't overwhelmed in a rich outer environment. E_{inner} is influenced not only by what the SCAs sense, but also by the CA's internal state. For instance, in a predator-prey simulation, if the predator is hungry and senses an animal, it would show up in E_{inner} as food. On the other hand, if the predator has just eaten, then the animal would appear as just another animal in E_{inner}.

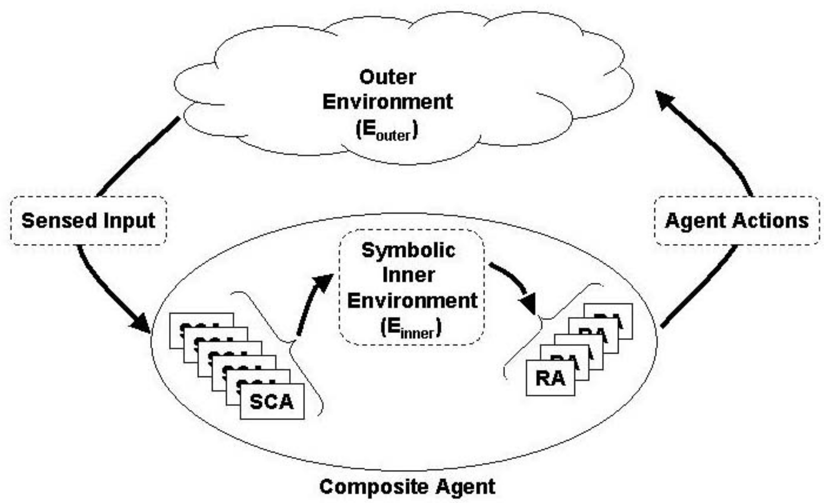


Figure 9. Composite Agent

The symbolic inner environment is the agent's perception of the shared outer environment within which it operates. E_{inner} has little resemblance to the actual outer environment, rather it is an encoding of E_{outer} optimized to suit the CA's specific function. The role of an SCA is not unlike the role of radio navigation aid used by a pilot. The navigation aid senses radio signals in the outer environment and converts them into directional information that the pilot can use to navigate the aircraft. The inner environment used by the pilot for making decisions has little resemblance to the view looking out the window, but it is optimized for use by the pilot in navigating the aircraft.

Combined with the SCAs is a set of *Reactive Agents* that operate on the symbolic inner environment and generate actions for the CA to perform. Each *Reactive Agent* has a set of possible goals and an apparatus for managing the process of selecting the active goal or goals.

D. REACTIVE AGENTS AND GOAL MANAGEMENT

Composite Agents contain a set of *Reactive Agents* (RAs), where each reactive agent is responsible for promoting a specific behavior of the Composite Agent. The set of RAs taken as a group, define the CA's set of high-level behaviors. The RAs operate within the world of the inner environment. They take as input sensory information from E_{inner}, and produce as output actions for the agent to perform.

Each RA has one or more goals specific to furthering its behavior or function. So at any given time there are numerous goals competing for the CA's attention. Just as humans have multiple goals (sometimes conflicting), an agent too can have multiple goals it wishes to satisfy. In human decision-making, goals are constantly shifting in priority, based on the person's context and state. Agents can mimic the flexibility and substitution skills of human decision-making with a variable goal management apparatus within the RAs. Thus, contextually appropriate, intelligent behavior emerges from this goal apparatus. RAs interpret the symbolic inner environment and through their goal apparatus, process this information to balance their goals and return an appropriate action for attaining their highest priority goal(s) (Figure 10).

The basic definition of a goal has four components: a state, a measurement method, a weight, and a set of actions for achieving the goal (Equation 3).

$$
\begin{gathered}
goal = < s, mm, w, \{a\} > \\
s = state \in \{inactive, active, achieved, \ldots\} \\
mm = measurement\ method \\
w = weight \\
\{a\} = an\ action\ set\ for\ achieving\ the\ goal
\end{gathered}
$$

Equation 3. Goal Definition

The goal's *state* is an indication of whether the goal is active, inactive, achieved, or in some other domain specific status. The *measurement method* translates the sensory input received by the RA into a quantifiable measure of the current strength of the goal and how well it is being satisfied. This permits an agent to prioritize goals and adjust goal states based on context. A goal may also have a *weight* attached that can be used to adjust the importance or priority of the goal based on experience. Tied to each goal is an *action* or set of actions for achieving the goal under varying circumstances. The end result is that within the RA goal apparatus there are multiple goals that are constantly changing – moving up and down – with the top (active) goals dominating the agent's behavior.

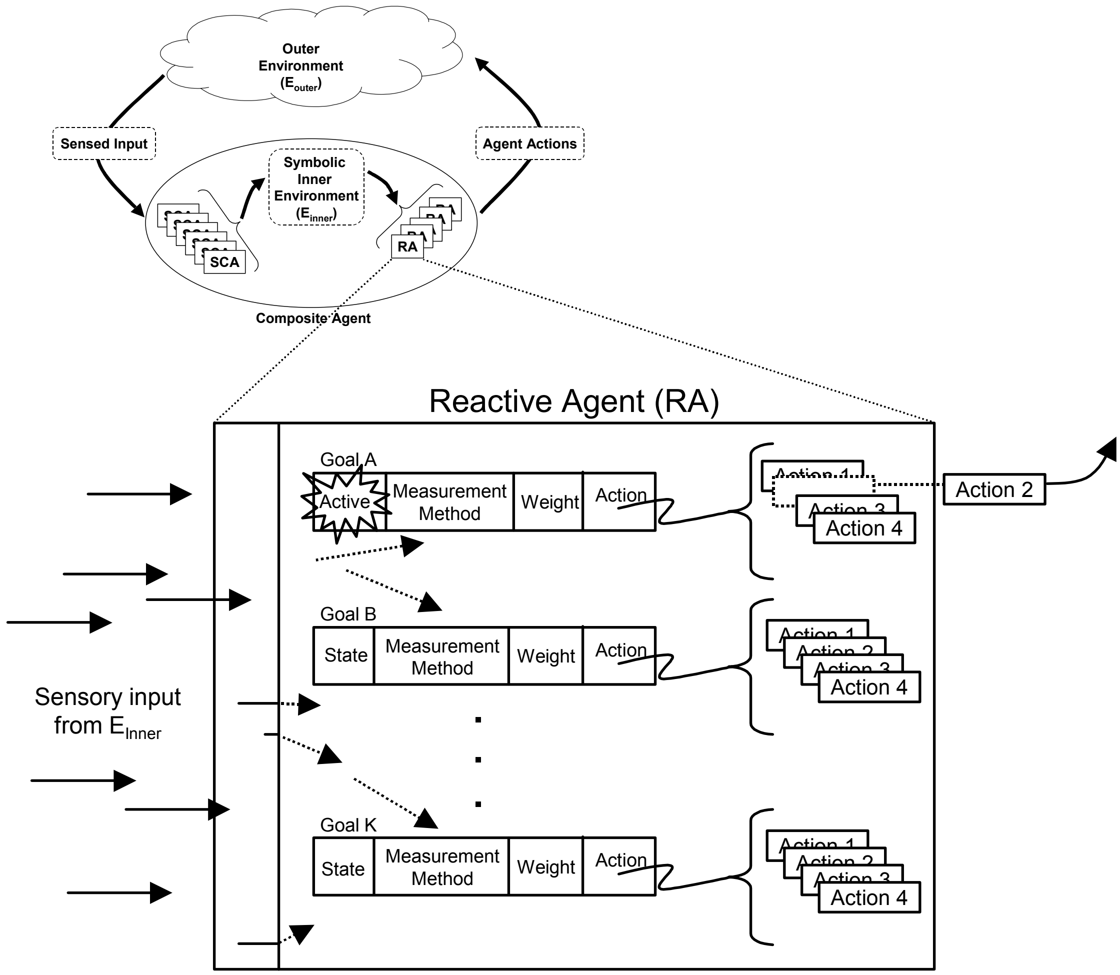


Figure 10. Reactive Agent

Additionally, agents can discard behaviors that do not further their goals, and increase the use of behaviors that have proved successful in reaching goals. This simple process serves as a reactive learning system where the agent learns from the environment, based on "what works," with no human expertise or intervention.

Goal switching based on a dynamically changing environment produces innovative and adaptive behavior. The ability to continually adapt to an ever-changing environment in real time is provided through a construct called *connectors*.

E. CONNECTORS

This section presents a formal definition of a *connector* and describes its fundamental behaviors. [VanPutte, 2002] extended the definition of a *connector* to develop *iconnectors* that are used as an inter-entity communications mechanism in the

information assurance domain. In the course of his work, VanPutte presents a Unified Modeling Language (UML) representation of *iconnectors* and their function. In addition, he describes a graphical notation for visualizing *iconnectors* that is used here to visualize *connectors*.

1. The Biological Inspiration Behind Connectors

Multi-agent simulations are used to model inherently complex systems. One of the major challenges is developing a communication protocol capable of standing up to the potentially combinatorial explosion in sensory information that floods the environment as the agents interact with each other and their surroundings. The protocol must be flexible enough to allow agents to define what information they need and when they need it. This is particularly difficult in agent-based modeling where the entities are autonomous and capable of changing their scope of interest to suit their current goals.

The complexity described above and associated communication requirements are not unlike that found in molecular biology when studying cell-to-cell communication, coordination and control. The following excerpt was taken from a molecular biology text and captures the fundamental role signaling plays in cellular biology.

> …the behavior of each individual cell in multicellular plants and animals must be carefully regulated to meet the needs of the organism as a whole. This is accomplished by a variety of signaling molecules that are secreted or expressed on the surface of one cell and bind to receptors expressed by other cells, thereby integrating and coordinating the functions of the many individual cells that make up organisms as complex as human beings. [Cooper, 1997]

The signaling mechanisms found in molecular biology are the foundation of a remarkable intracellular and intercellular coordination and control system that manifests itself in the form of a functioning human body. This system serves as the inspiration behind a software agent communication mechanism called *connectors*. The remainder of this section presents an overly simplified, and primarily pictorial introduction into the signaling mechanisms used for cellular communication.

All cells receive and respond to signals from their surroundings. They respond to signaling molecules secreted by other cells, allowing cell-to-cell communication. Some of the molecules carry signals over long distances, while others act locally to convey

information between neighboring cells. There are four major types of signaling. Signaling over long distances occurs through Endocrine signaling, where molecules are secreted by cells and carried through the circulatory system to target cells a great distance away (Figure 11A). In Paracrine signaling, the released molecules act on neighboring target cells (Figure 11B). Contact-dependent or direct cell-to-cell signaling occurs when the signaling molecule remains attached to the signaling cell (Figure 11C). Finally, Autocrine signaling takes place when a cell produces a signaling molecule to which it also responds (Figure 11D).

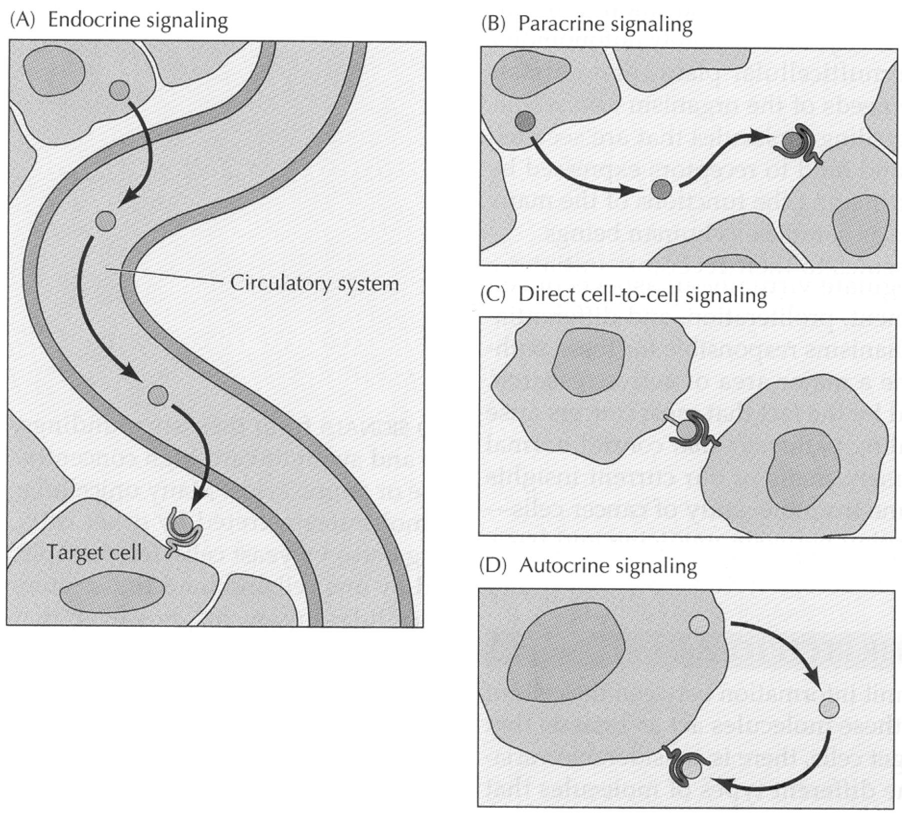


Figure 11. Cell Signaling Methods from [Cooper, 1997]

Cells in multicellular organisms are typically exposed to hundreds of different signals in an environment that can act in millions of combinations. The cell must respond to this plethora of signals selectively, according to its own specific character (Figure 12).

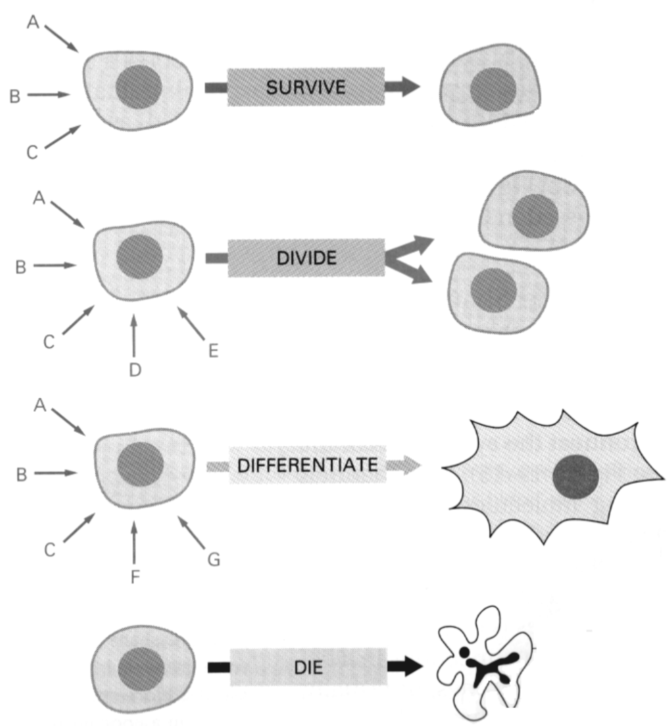


Figure 12. Cells Responding to Combinations of Extracellular Signals from [Alberts *et al.*, 2002]

A cell may be programmed to respond to one combination of signals by differentiating, to another combination by multiplying, and yet another by performing some specialized function. The hundreds of signal molecules that animals make can be used to create an almost unlimited number of signaling combinations. The use of these combinations to control cell behavior enables an animal to control its cells in highly specific ways by using a limited diversity of signal molecules [Alberts *et al.*, 2002]. In other words, from a limited set of signals, it is possible to control extremely sophisticated and specialized behavior.

One way this is accomplished is through signal cascading. Figure 13 shows how an extracellular signal molecule can trigger a cascade of intracellular signals. The extracellular signal molecule at the top of the diagram binds with a receptor protein extended from the cell membrane. This single external interaction begins a signal-response process that results in gene transcription occurring within the nucleus of the cell.

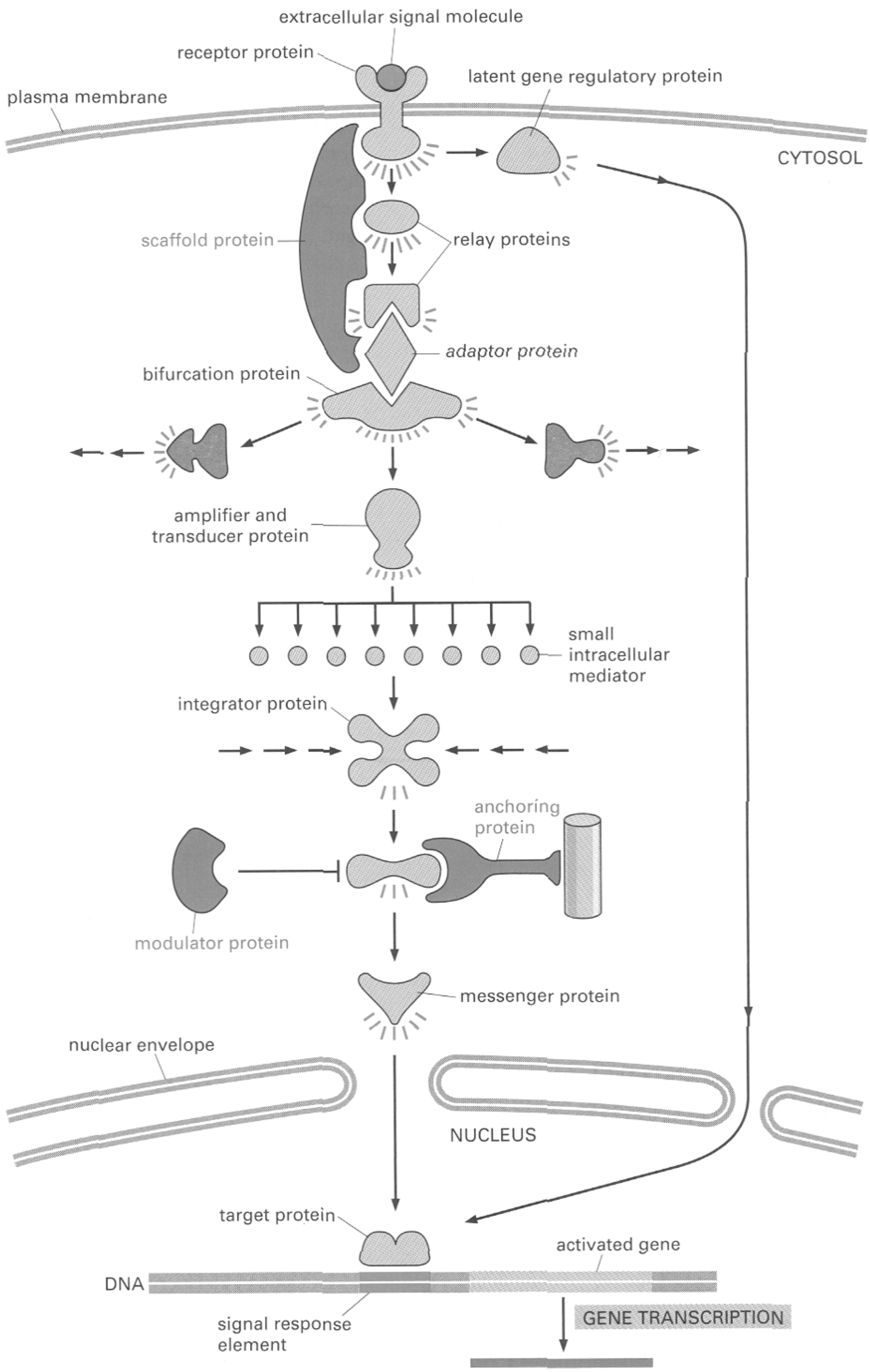


Figure 13. Intracellular Signal Cascading from [Alberts *et al.*, 2002]

This section provided a brief description of cellular signaling to demonstrate how a system of signals and receptors can be used to control complex behavior. The next section describes a software construct called a *connector* that is used to mimic certain

aspects of cellular signaling. *Connectors* are not intended to duplicate exactly each of the cell signaling methods; rather they are an abstraction that captures the general nature of the distributed communication and coordination protocol.

2. Connector Definition

Connectors are a software component that can be "bolted on" to agents and objects in a multi-agent simulation to provide a communication and coordination capability. The base set of connector behaviors can be augmented for the specific simulation to provide a powerful and flexible domain specific communications protocol. Through a process called *connecting,* agent-to-agent and agent-to-object *connections* are established during which two-way communication occurs between the *connected* entities. Connectors are active objects that sense and react to the environment. They activate (extend) and deactivate (retract) in response to changes in the state of the *entity* to which they are attached. As the entity's state and the state of the environment changes, the connectors sense the changes and extend or retract accordingly.

Equation 4 defines a connector in terms of seven components. Connectors are attached to a *host* entity and have an associated *control function* that defines its behavior. Connectors, at a minimum, implement four basic *actions*: extend, retract, connect and disconnect. These actions are used to modulate the connector between its three primary *states* of extended, retracted and connected. Connectors are defined by *type* and can extend in one of two *modes*: receptor and stimulus. Associated with each connector type, is a set of possible *values* the connector can assume. As a receptor, the connector is capable of connecting with a connector of a matching type that is extended in stimulus mode. That is, the connector is capable of attaching to a signal (molecule) released by an agent or object in the system. In addition to the basic actions, connectors may be augmented with type-specific actions.

$$
\begin{array}{c}
\textit{Connector } c = \langle h, f_c, s, m, t, V, v, A \rangle \\
h - \textit{host} \\
f_c - \textit{control function} \\
s - \textit{state; } s \in \{\textit{extended, retracted, connected}\} \\
m - \textit{mode; } m \in \{\textit{receptor, stimulus}\} \\
t \text{ - } \textit{type} \\
V - \textit{set of possible values for connector type } t \\
v - \textit{current value } (v \in V) \\
A - \textit{action set;} \\
A = \{\textit{extend, retract, connect, disconnect, type-specific actions}\}
\end{array}
$$

Equation 4. Connector Definition

a. Graphical Notation

Before proceeding further, it is necessary to briefly describe the graphical notation developed in [VanPutte, 2002] for visualizing connectors. The notation was developed to depict a specific type of connector (*iconnector*) and has been modified somewhat to present a general definition of *connectors*. Figure 14 depicts three connectors in various states; retracted, extended (stimulus) and extended (receptor). The solid lines extending to the left into environment *E* indicate the source of the input to the respective control functions. All three connectors are receiving input from various parts of environment *E*. The symbol to the right side just below the connector arm indicates the connector *type*. Connector W is of type γ and connectors X and Y are type β. Connector W is shown in a *retracted* state, while X and Y are *extended*. When extended, connectors operate in one of two modes, *stimulus* or *receptor* as indicated by the different ends on connectors X and Y. The symbol in the circular end of connector X indicates the *value* of the connector. The receptor on connector Y is labeled with the set of values it can connect with (a and c); in the case where it can connect with any value, it is labeled with an ‘∗’.

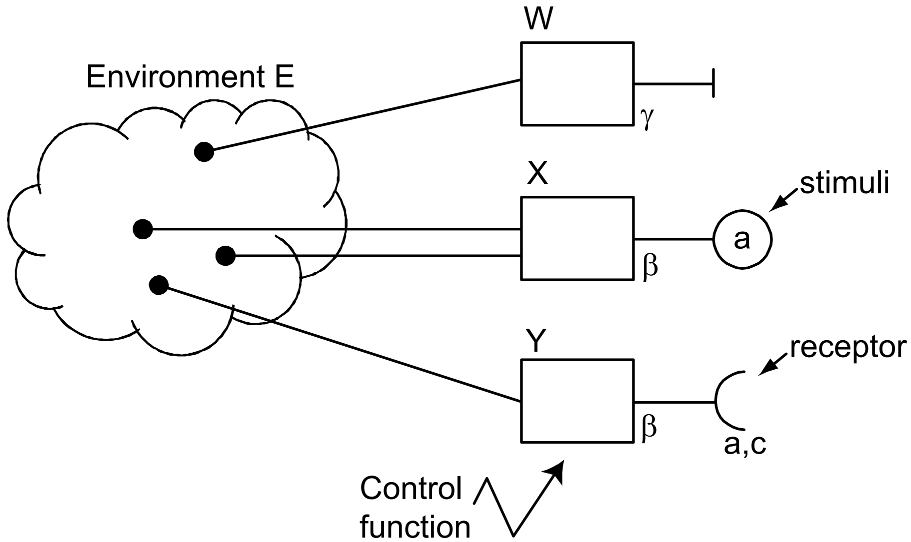


Figure 14. Graphical Depiction of Connectors

b. Host Entity

Connectors are attached to objects and agents in the simulation. The object or agent to which they are attached is called the host entity. As such, connectors become an extension of the host and function to describe the host's state. The host entity and connector control function are tied closely together in that the host entity serves to define the scope of information available to the connector's control function. That is, the control function operates primarily on input from the host. However, when the connector is extended, information received via the connector is also available. Figure 15 shows a connector attached to a host agent. The control function is receiving input from the agent's inner environment. With the connector extended into the outer environment in receptor mode, any stimuli received by the connector are also fed to the control function. Connectors are also associated with an agent modeling construct called tickets, which is described later in this chapter.

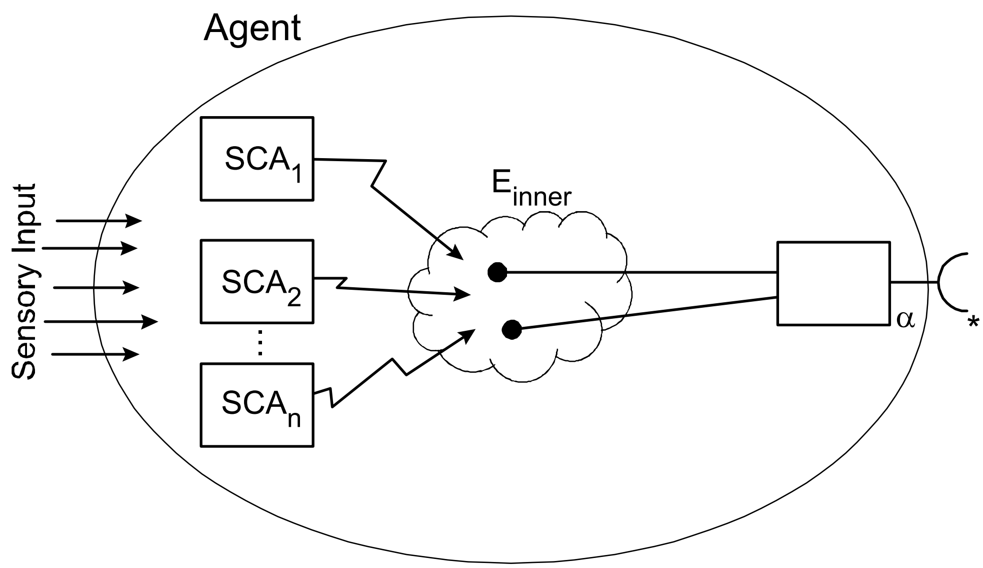


Figure 15. Connector Attached to a Host Agent

c. Control Function

The control function (f_c) manages the connector's state changes, as well as value changes, and controls execution of domain-specific actions. The term *behavior* is used in the context of connectors to describe "the execution of state changes, value changes and domain-specific actions relative to the state of the host to which it is attached."

The control function defines the behavior of the connector in terms of domain-specific measures and values. For instance in the case of state changes, f_c defines precisely, based on the information available to the connector, when the connector changes state. Figure 16 shows two agents from a predator/prey simulation where both agents are equipped with food connectors.

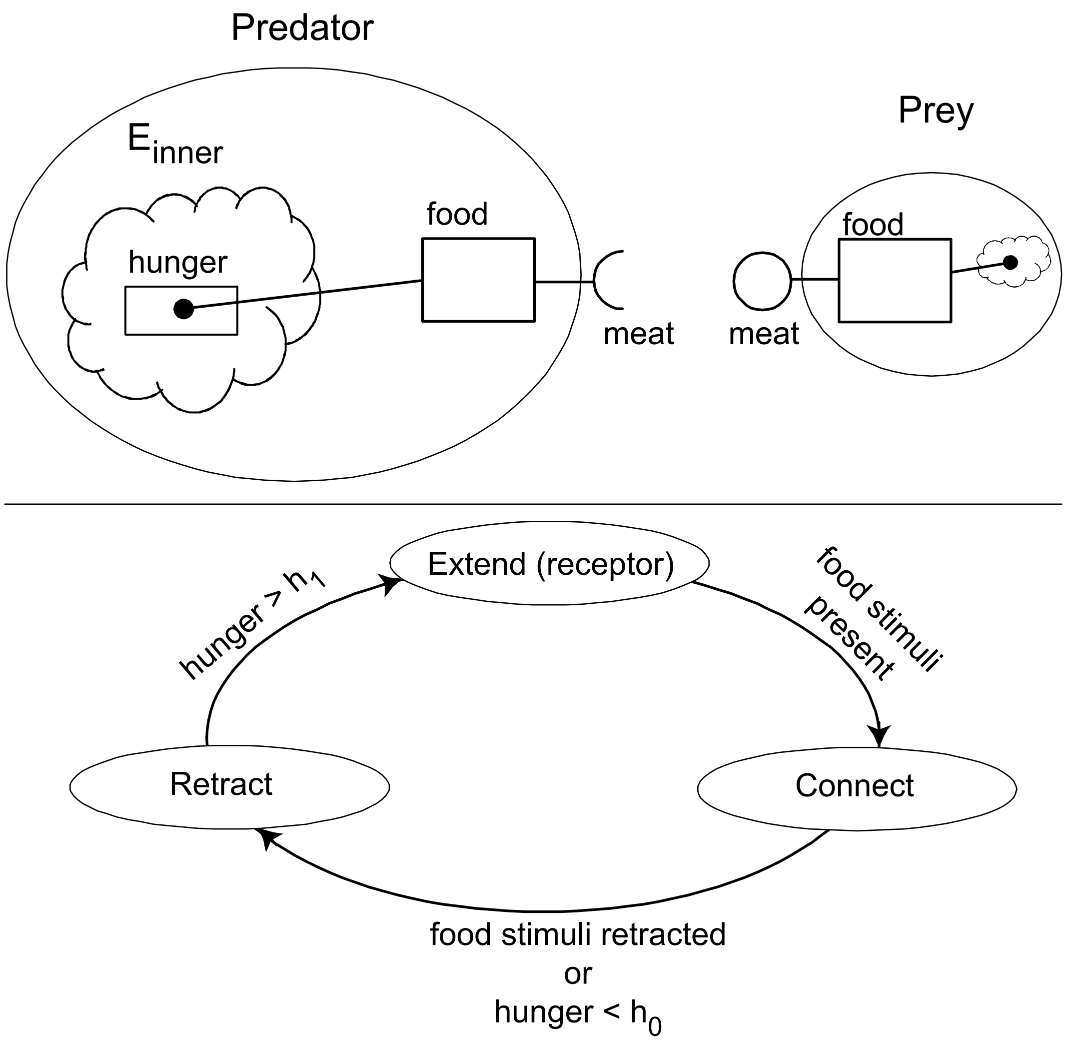


Figure 16. Predator/Prey Control Function

The predator's inner environment includes a state variable that captures its hunger level. The control function of the food connector is linked to this hunger state variable. As the hunger level increases above a threshold of h_1, the connector extends in receptor mode in an attempt to locate food. If the predator senses the presence of food via a food stimulus, then a *connection* is established. Because of the *connection*, the predator is able to satisfy its hunger. The *connection* is broken once the prey's food source is exhausted (connector retracts), or the predator's hunger level is reduced below a threshold of h_0.

d. State

A connector has three possible states: extended, retracted and connected. In addition, an extended connector has two modes, receptor and stimulus. The control function manages the state transitions as depicted in Figure 17.

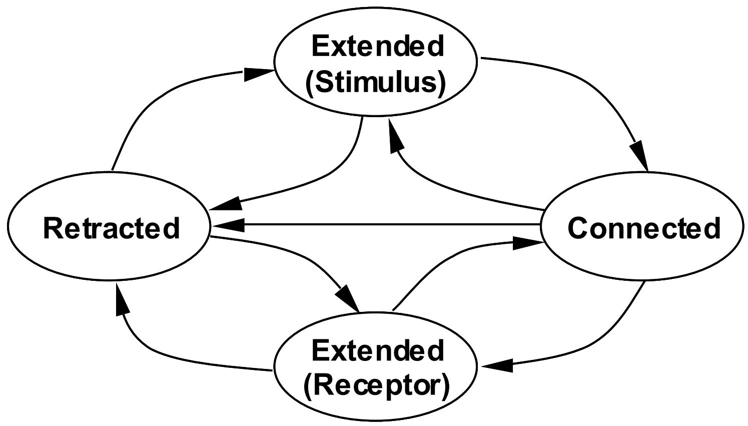


Figure 17. Connector State Transition Diagram

(1) Retracted. In a retracted mode, the connector is inactive with respect the environment outside of its host entity. The control function continues to receive updated information from within the host, and uses this to change the connector's state. From a retracted state, the connector is able to transition to an extended state, either in stimulus or receptor mode.

(2) Extended. When a connector extends, it does so in one of two modes: receptor or stimulus. A full description of the two modes is provided below. When in an extended state, the connector has access to the environment outside of its host. In the stimulus mode, the connector is broadcasting its value to all connectors extended in receptor mode capable of sensing the stimulus. In receptor mode, the connector senses the environment for stimulus from connectors of a compatible type. From an extended state, a connector can transition to a retracted state, or alternatively, to a connected state.

(3) Connected. Connectors extended in receptor mode are capable of establishing a connection via a *connect* action. A connection opens a conduit between the connected entities through which high-level interactions are possible. Examples include pushing and pulling data, increasing or decreasing an agent's resource levels or

adding procedural knowledge to an agent to simulate an increased level of training. In [VanPutte, 2002] once an agent establishes a connection with piece of infrastructure, the agent can "pull" information (i.e. access codes and passwords), push information (i.e., a virus), or directly modify the entity (wipe the hard-drive). As can be seen, the action set is specific to the connected entities. The type and scope of action is application dependent.

Connectors in a connected state can break the connection with a *retract* action and transition to a retracted state, or execute a *disconnect* action and return to an extended state in receptor or stimulus mode.

e. Mode

When connectors extend, they do so in one of two modes: *stimulus* or *receptor*. In *stimulus* mode the connector takes on a single *value* from its set of possible values and broadcasts its *type*, *value* and *host* entity to the environment. From the stimulus perspective, the information is sent with no specific intended target.

Connectors extended in *receptor* mode are essentially in a listening mode. They listen for, or sense, connectors in their environment of the same type that are broadcasting in stimulus mode. Incoming sensory information, in the form of *value* and *host* entity, is passed to the control function for processing.

While in *receptor* mode, it is possible for the connector to enter into a *connected* state with another connector. The receptor is *labeled* with a set of values from the connector's value set that the *receptor* is capable of connecting with. This set is dynamic in that the connector's control function continually updates the members. Once *connected*, additional high–level interactions are possible between the connected entities. These "high-level" interactions are defined by the host entities. The connectors simply open up the communications channel between the hosts.

Equation 5 defines a current value function (Φ) that returns the set of values the receptor can connect with, or the value of the stimulus.

$$\text{Given, } c = \langle h, f_c, s, m, t, V, v, A \rangle$$

$$\Phi(c) = \begin{cases} V_c \subseteq V & ((s = extended)\ \&\ (m = receptor)) \\ v \in V & ((s = extended)\ \&\ (m = stimulus)) \\ undefined & otherwise \end{cases}$$

$(V_c$ = *the set of values labeling receptor c)*

Equation 5. Connector Current Value Function (Φ)

f. Type

Connectors are defined in terms of their type. Only connectors of the same type can connect with one another. The type designations for connectors are application dependant; there are no predefined types. In the story engine instance described in Chapter VII, character agents are defined with seven personality traits and six resource categories, with a corresponding connector type for each. This finite number of traits is based on the particular application – Army careers/values – and is not a limitation of the story engine. Defining the connectors based on personality and resources establishes a character-to-character communications protocol structured in terms that are important to Army career progression stories.

g. Value

For each connector type, there is a defined range of possible values. When a connector is extended in stimulus mode, the connector's current value is broadcast along with the *type* and *host* entity. Much like *type*, *values* are application dependent with no predefined *values* for connectors. In the story engine characters described above, one of the resource categories is *Energy*. For the *Energy* type connector, there is a set of five possible discrete values ranging from *Low* to *High* (Figure 18). In the example, the connector value is from a discrete set but there is nothing to prevent the connector value from being the output of a continuous function.

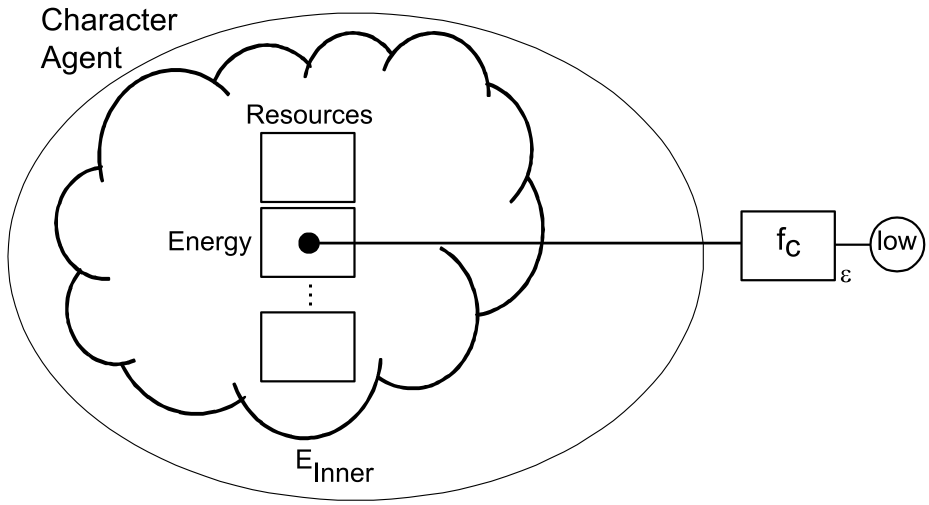


Figure 18. Character Agent Energy Connector

h. Action set

The action set defines the actions that the connector's control function can initiate. There are four basic actions required of a connector: *extend*, *retract*, *connect* and *disconnect*. In addition to the four basic actions, connectors may be extended with type-specific actions appropriate for the given simulation domain. The basic actions are used to transition the connector between its states (Figure 17).

Extend and *retract* expose and rescind the connector arm as previously described. If the connector is in a connected state, then *retract* breaks the connection. *Disconnect* breaks the connection between two connectors, but leaves the connector extended. *Connect* initiates a connection between two connectors provided all of the prerequisites have been met. Equation 6 defines the predicate $conn(c_1, c_2)$, which describes the requirements for two connectors to *connect*.

$$
\begin{gathered}
\textit{Given,} \\
c_1 = \langle h_1, f_{c1}, s_1, m_1, t_1, V_1, v_1, A_1 \rangle \\
\textit{and} \\
c_2 = \langle h_2, f_{c2}, s_2, m_2, t_2, V_2, v_2, A_2 \rangle \\
\textit{then } c_1 \textit{ can connect to } c_2 \ (conn(c_1, c_2)) \\
\textit{iff:} \\
s_1 = s_2 = extended \\
m_1 = receptor \ \&\ m_2 = stimulus \\
v_2 \in \Phi(c_1) \\
t_1 = t_2
\end{gathered}
$$

Equation 6. Requirement for Two Connectors to Connect – conn(c_1, c_2)

At a minimum, for a connection to be established, the receptor and stimulus connectors must be of compatible types and the stimulus value must match the label on the receptor. Once these minimums are met, the control function *may* execute a *connect* action binding the two entities together. It is important to note that the connection is not necessarily automatic. Even though a connection is possible, f_c must still initiate the *connect* action. As an example, in the case of *iconnectors* from [VanPutte, 2002], f_c is directed to initiate the connection by a higher level mechanism called an *ibinder*. In the story engine CMAS described in Chapter VI, connections are initiated locally by the agents.

F. TICKETS

One of the major benefits of agent-based modeling is the innovative and adaptive behavior agents bring to simulations, however, it is often desirable to balance this with doctrinally correct and appropriate actions. Symbolic Constructor Agents and the goal apparatus were developed to control the agent's sensory capability and decision-making. In order to provide agents with a rich procedural knowledge base while still supporting adaptive behavior, a data structure called *tickets* has been developed. Tickets allow agents to apply procedural knowledge in context. They define the agent's action set, i.e., its means to achieve its goals. They are used to organize procedural knowledge and provide the ability to balance doctrinal behavior with adaptive, innovative action, resulting in enriched problem solving behavior.

Tied to each of an agent's goals are one or more tickets that define how to achieve the goals (Equation 7). Tickets are defined by a control function (f_c), a set of connectors (C), and a set of frames (F).

$$\begin{array}{|c|}\hline
\textit{Ticket} = < f_c,\ C,\ F > \\
f_c \textit{ - control function} \\
C \textit{ – connector set} \\
F \textit{ - } \{frame_1, frame_2, \ldots frame_n\}; \\
frame_i \in \{action, ticket, reference\} \\ \hline
\end{array}$$

Equation 7. Ticket Definition

1. Ticket Control Function

The control function has four jobs. When the ticket is first executed, it performs any necessary initialization. Second, f_c controls the sequence and manner of frame execution. Since f_c is user defined, any execution sequence is possible, including sequential, random, looping, etc. In addition, the tickets can be defined so they execute through their entirety as a single action (execute all frames without interruption), or they can execute a single frame each time it is the agent's turn to act (single step). Third, when the ticket is complete, f_c resets the ticket as appropriate. Finally, it coordinates with the attached connectors to initiate their *connect* actions.

2. Ticket Frames

Tickets are comprised of a set of frames, F. A frame can be thought of as a container that holds a procedural step. However, simply encoding step-by-step procedural knowledge and linking it to various goals is not sufficient for creating intelligent behavior. The desire is to apply the most appropriate actions or procedures for the "given situation." In a dynamic system, the "given situation" not only changes constantly, but is often so complex the system designer cannot conceive of, much less account for, every possibility. Therefore, the mechanism for determining the "most appropriate" procedures must be flexible and able to support the same level of complexity as the changing contexts of the dynamic system. By combining connectors with tickets, the desired flexibility can be achieved. Figure 19 is a graphical depiction of a connector-based ticket.

Ticket frames can be static or dynamic [Hiles *et al*., 2001], [VanPutte, 2002]. Static frames hold a "by-name" call for a specific ticket or action, while dynamic frames

hold a more general "reference" for the type of ticket or action. By-name specification of tickets and actions equates to hard-coding behaviors at design time. In many cases, constraining an agent's action in this way is desirable. In Figure 19, frames one and three contain by-name calls to *Action X* and *Ticket A* respectively.

Ticket frames can also hold ticket and action *references*. These *references* allow the designer to specify the general nature of the behavior, but delay the final binding of the specific action or ticket until run time.

References are simple lightweight objects that coordinate the actions of connectors extended in receptor mode. When it comes time for a dynamic frame to execute, the reference object initiates a connection with an appropriate ticket or action. This run-time binding ensures the behavior will be appropriate for the given situation as perceived by the agent. A detailed description of *references* is provided with the discussion of Composite Agent actions in Chapter V.

Returning to Figure 19, frame two contains a *reference* for an action described by a type α connector with value r and a type β connector with value s. These extended connectors (type α and β), as well as their respective values (r and s), were not fixed at design time, rather they result from the current state of the host agent. They are not bound until the reference is evaluated for execution at run-time. In this example, *Action Y* meets the criteria and will be executed in frame two. Similarly, frame four contains a reference for a ticket with a type α connector with any value ($*$). *Ticket B* will be executed at frame four.

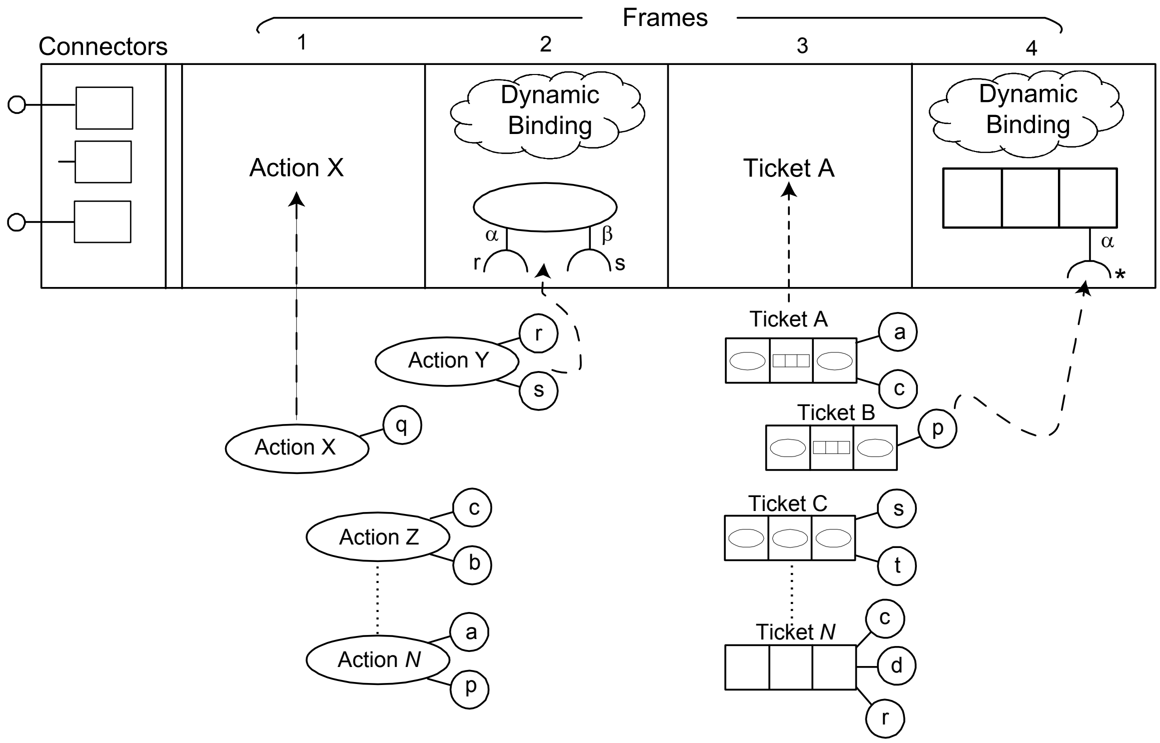


Figure 19. Example Connector-based Ticket

3. Ticket Connectors

Tickets have prerequisites that must be met in order to activate. The prerequisites are defined in terms of connectors and are specific to each ticket. Figure 20 shows a view of a composite agent with the connectors' control functions receiving input from the agent's inner environment. Perceptions from the outer environment are processed by the SCAs which in turn update E_{inner}. Connectors attached to the tickets are likewise updating based on the agent's inner environment and dynamically change state as E_{inner} changes. In Figure 20, *Action X* has the necessary connectors extended in stimulus mode that will allow it to connect with the reference in frame one of *Ticket A*.

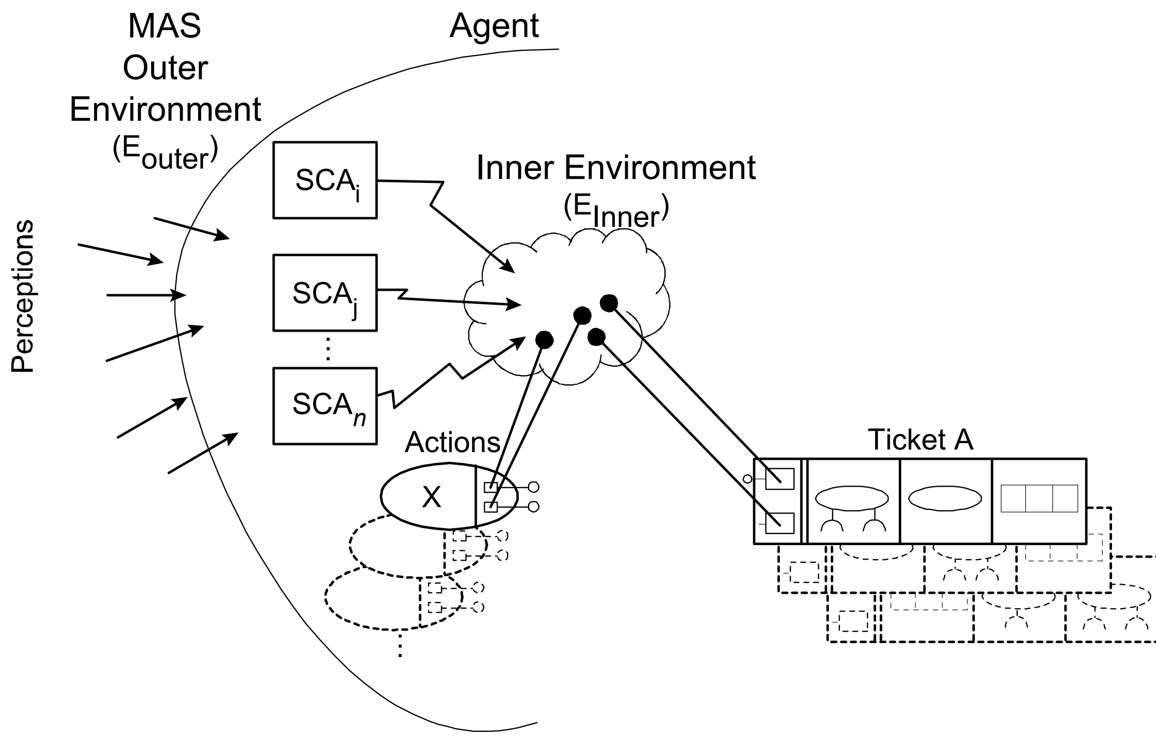


Figure 20. Ticket and Action State Dynamically Maintained with Connectors

With the connectors continually reacting to the environment, behavioral and procedural knowledge (tickets and actions) can bind at run-time to fit the context as it develops. This binding is based not only on the state of the environment, but also on the goals of the agent, its capabilities, and its social interactions with other agents. In this way, the correct procedural knowledge can be brought to bear in the appropriate situation.

G. SUMMARY

In this chapter, the concept of semi-fluid software structure was introduced. Connectors and tickets were formally defined and the biological inspiration behind connectors was presented. When connectors are combined with tickets and actions, *reference* actions can be used to bind actions to tickets at run-time. This run-time binding provides agents with the ability to execute contextually appropriate actions in pursuit of their goals. Connectors, tickets and composite agents form the foundation for a Connector-based MAS (CMAS) architecture. The following chapter defines the CMAS architecture and develops a simulation model that makes extensive use of connectors and

the process of connecting, by which agents employ a simulation specific “best-fit” algorithm to bind to other agents.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONNECTOR-BASED MULTI-AGENT SYSTEM ARCHITECTURE

A. INTRODUCTION

This chapter introduces the Connector-based Multi-Agent System (CMAS) architecture. A formal definition of a CMAS is presented, along with an architecture for a connector-based composite agent. The CMAS architecture and connector-based agents form the foundation for the story engine.

B. CONNECTOR-BASED MULTI-AGENT SYSTEM

One of the difficulties in modeling complex systems is defining the function described in Equation 2 that reduces the set of all possible actions that an agent can take (Ω_a) to a set of actions that are appropriate for the current situation ($\Omega_{a,t}$). The function is far too complex for a top-down design. The decision of what to do next, based on all of the possible things that can be done, must be pushed down to the local level where the agent decides. In addition, once the decision is made to act, it is preferable to not simply take any action that will work, but act in a manner that is most appropriate to the given situation. In order to take the appropriate action, binding the specific action should be delayed as long as possible.

Connectors provide a general modeling construct used to facilitate communications and assist in the delayed binding of actions. Their design makes them an excellent mechanism for representing the "current situation," and when combined with tickets, they provide the ability to express "what should be done" with the flexibility of delaying the decision of "how it should be done."

In a connector-based MAS, the environment is populated with agents and objects that express their state through extended connectors. The extended connectors provide a sensory stream for other agents in the simulation. Through an operation of connecting, agents interact with one another and objects in the environment. Connecting employs an application specific "best-fit" algorithm to bring entities together and facilitate their interaction. These connections are active for a finite period of time during which the connected entities interact through the communication channels opened up by the bound connectors. Through a continual process of connecting and disconnecting, the system

evolves (Figure 21). In the case of the story engine, this process results in stories being generated one connection at a time.

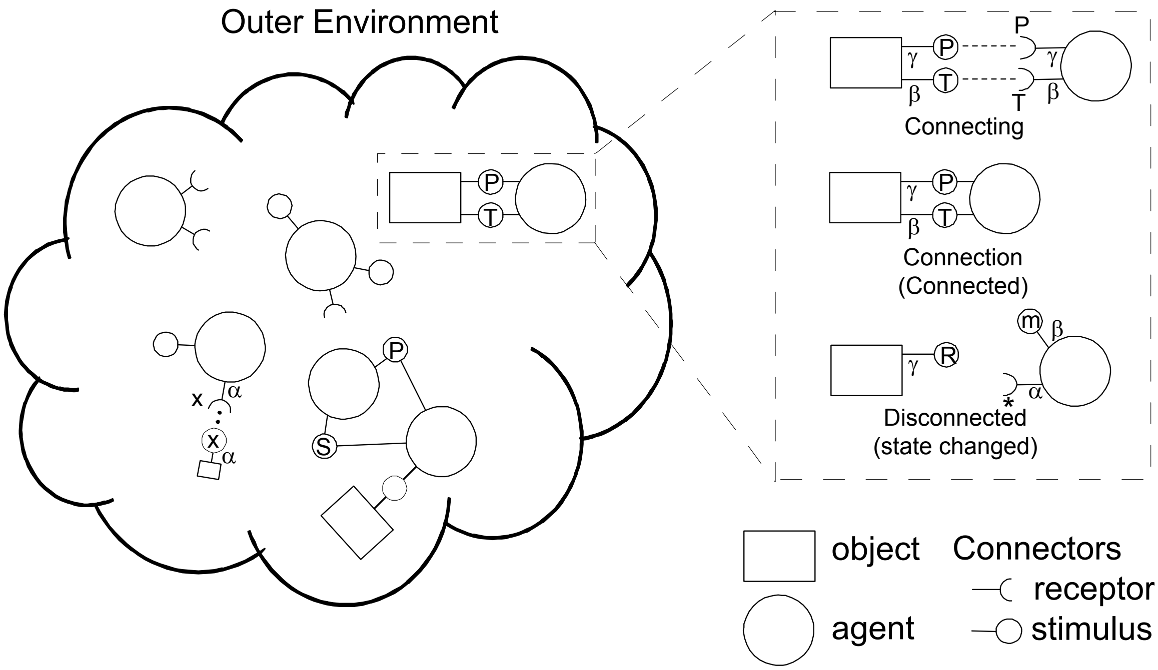


Figure 21. Connector-based MAS

Equation 8 defines a MAS architecture based on the fundamental behaviors of connectors and tickets.

$$
\boxed{
\begin{array}{c}
MAS = \{E, O, A, R, Op, Laws, C\} \\[4pt]
E - Environment\ \ = <A, O, C> \\
O - Objects\ situated\ in\ the\ environment \\
A - Connector\text{-}based\ agents \\
R - Relations\ linking\ agents\ and\ objects \\
Op - Operations \\
Laws - Laws\ governing\ the\ environment \\
C - Connectors
\end{array}
}
$$

Equation 8. Connector-based Multi-Agent System (CMAS) Definition

C. ENVIRONMENT

As with a MAS defined by Equation 1, a CMAS may be situated or non-situated. The environment is populated with agents and objects, and associated with each of these entities are connectors that describe the entity's current state. In a CMAS, the agents do not directly perceive the other agents and objects in the environment; they sense the connectors that are exposed to the environment by the entities. Therefore, while

Equation 9 includes agents and objects in the environment, they only exist through their connectors.

$$E = \langle A, O, C \rangle$$

$$A - Agents$$
$$O - Objects$$
$$C - Connectors$$

$$C = c(A) \cup c(O)$$
$$\text{where } c(x) \text{ is the connector set associated with entity } x;\ x \in A \text{ or } x \in O$$

Equation 9. CMAS Environment

D. OBJECTS

Objects in a CMAS are non-agent entities in the environment. They can be perceived, created, destroyed and modified by agents. Unlike Ferber's definition of a MAS, agents are not a subset of the objects. The primary difference between agents and objects is that agents are active entities capable of taking action based on their own intent.

Objects are differentiated by type, and within the type, further differentiated by specific instances of the type. For example, rocks, trees and radio transmitters are object types. For each of these types, there are instances with attributes specific to the instance. In the case of radio transmitters, broadcast frequencies might be used to differentiate transmitter instances.

CMAS objects depicted in Figure 22 and defined by Equation 10 consist of connectors representing their state, and a control function that allow the objects to change state based on interactions with agents in the environment. In the case of objects, the control function f_c is basically a state machine; on input x, the object reacts in a predictable way, resulting in new state y.

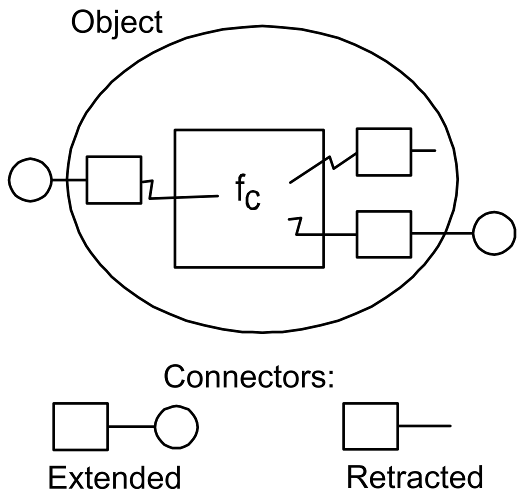


Figure 22. CMAS Object

$$o = \langle f_c, C\rangle$$

f_c –function controlling the object's state
C – set of connectors defining the state of object o

Equation 10. CMAS Object Definition

In general, the set O is defined as the union of all of the possible object types for the system (Equation 11).

If there are n unique object types in the system, and
o_i = instances of object type i
then

$$O = \bigcup_i o_i \; ; \; i = 1,..,n.$$

Equation 11. CMAS Object Set (O)

Since objects can be created and destroyed during the life of the system, to be able to evaluate the state of the objects as a whole, O must be defined relative to the specific object instances at time t (Equation 12).

If there are n unique object types in the system, and
$O_{i,t}$ = {instances of object type i in the system at time t}
then

$$O_t = \bigcup_i O_{i,t} \; ; \; i = 1,..,n$$

Equation 12. CMAS Objects at Time t (O_t)

E. CONNECTOR-BASED COMPOSITE AGENTS

Connector-based agents are agents that adhere to the use of connectors as a primary communications mechanism and means of expressing their state to other agents in the environment. Many classes of intelligent agents can be employed in a CMAS; the only defining factor being that they interface with the outer environment through connectors. They do not have to employ connectors as their sole interface, but in as much as the agent wants its state to be known by other entities in the simulation, it does so through connectors. This section describes connector-based agents in terms of the composite agent architecture described in Chapter IV.

The previous chapter defined connectors and described their behavior. This section describes how to put connectors to work to create a flexible communicating agent. Equation 13 defines a connector-based composite agent.

$$Agent\ a = \langle A_{SC}, T, \Omega, E_{inner}, C_i, C_e, G, A_R \rangle$$

$$\begin{gathered}A_{SC} - set\ of\ symbolic\ constructor\ agents\\ T - tickets\\ \Omega - agent's\ action\ set\\ E_{inner} - agent's\ inner\ environment\\ C_i - local\ set\ of\ connectors\ internal\ to\ agent\ a\\ C_e - connectors\ used\ to\ externally\ portray\ agent\ a's\ state\\ G - set\ of\ goals\ for\ agent\ a\\ A_R - set\ of\ reactive\ agents\ to\ manage\ agent\ a's\ goal\ set\end{gathered}$$

Equation 13. Connector-based Composite Agent Definition

Composite agents employ a *sense-update-act* model where they sense their environment, update their internal view of the world, then decide what action(s) to take. The agent's sensory capabilities are defined in terms of its SCAs and extended connectors, which maintain an internal representation (E_{inner}) of the agent's view of the world. From this E_{inner} view, the agent employs a dynamic goal structure managed by reactive agents (RAs) to act in accordance with contextually appropriate procedural knowledge (tickets). In the course of their *sense-update-act* cycle, the agents make extensive use of connectors to maintain the state of their inner environment (C_i), select and take action, and express their state to the outside world (C_e). Figure 23 depicts a single composite agent processing sensory input from the outer environment. The following sections describe the connector-based composite agent in detail.

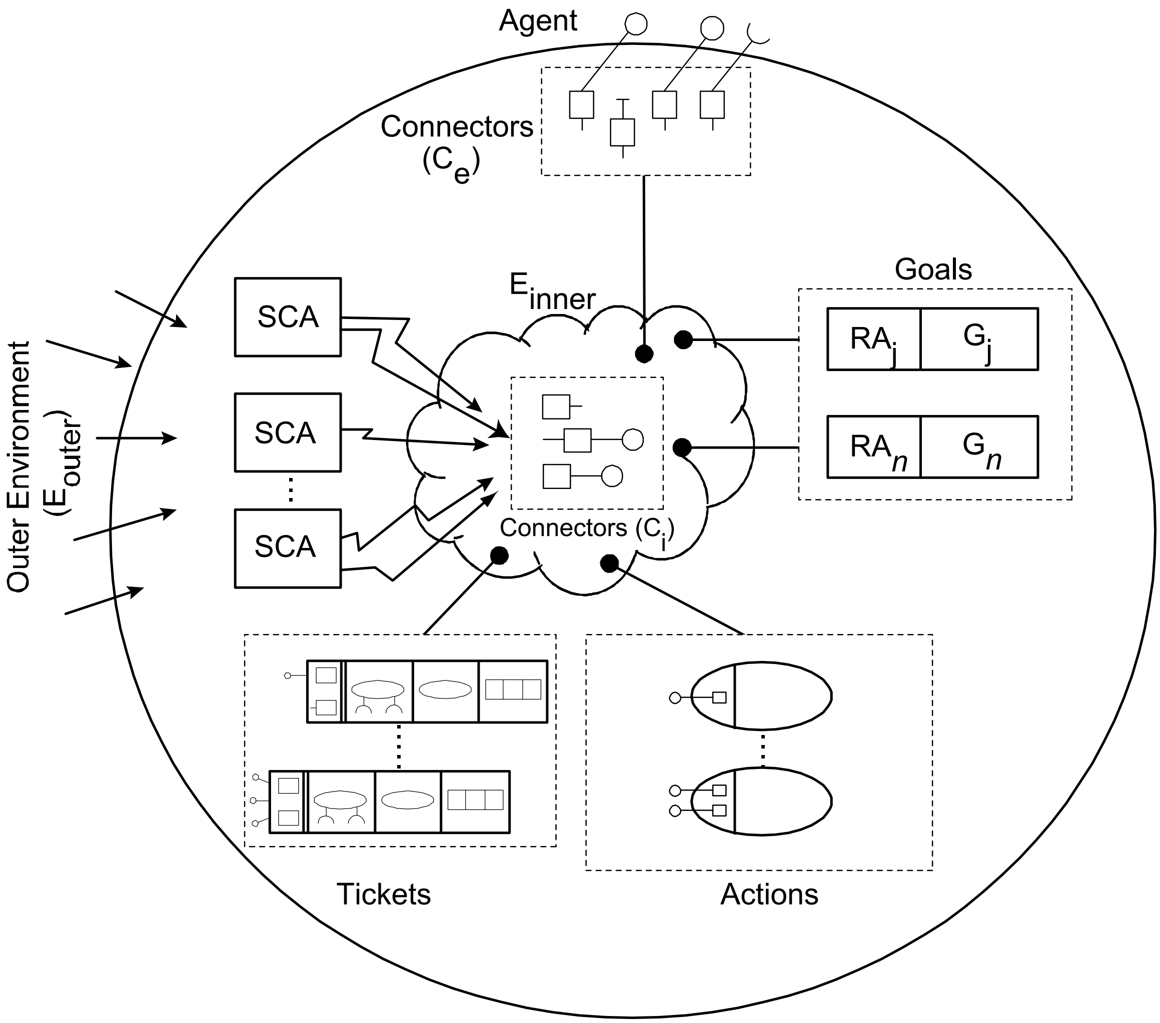


Figure 23. Connector-based Composite Agent

1. Symbolic Constructor Agents and E_{inner}

Symbolic constructor agents (SCAs) process sensory input from the outer environment (E_{outer}) and manipulate it as necessary to construct and maintain the agent's inner environment (E_{inner}). SCAs receive their stimuli by any available means; this may include the use of connectors, but it is not required. Depending on the particular domain being modeled, it might not be suitable for all external sensory information to be connector-based.

The architecture of SCAs is domain dependent and closely integrated with the structure of the inner environment. They can be simple reactive agents or complex cognitive agents using symbolic reasoning to interpret and correlate sensory streams to

build high-level representations of the input. SCAs can take advantage of any desired means to process sensory input and construct an inner representation.

As difficult as it is to define a precise architecture for SCAs, it is just as difficult to develop an exact definition of what comprises an inner environment. In general, E_{inner} can normally be described in terms of internal connectors, state variables and objects (Equation 14).

$$E_{inner} = \langle C_i, Sv, O \rangle$$

C_i – set of connectors internal to the agent
Sv – set of domain specific state variables
O - Objects

Equation 14. Agent Inner Environment (E_{inner})

The connector set C_i provides an intra-agent coordination and control capability. These connectors are identical to external connectors, with the exception that internal connectors never communicate directly with the outer environment.

State variables provide a measure of agent characteristics that are best captured through a numeric value. These values are updated through the SCAs and agent actions. Internal and external connectors often use these state variables as input to their control functions.

The objects of E_{inner} follow the definition from Equation 10. Their presence and state is broadcast via their extended connectors. Just as agents in the outer environment can connect to objects in their environment, SCAs and RAs can likewise connect to objects from E_{inner}.

2. Tickets and Actions

Composite agents act on their inner and outer environments through actions selected by a dynamic goal structure. Connector-based composite agents make use of connectors, in conjunction with a goal structure, to control and coordinate the application of both tickets and actions to achieve complex behavior. Tickets, by their design, make extensive use of connectors (Equation 7). Their function with respect to connector-based agents is the same as described in section IV.F.

Run-time binding of tickets and actions is accomplished through a *reference* action. References were introduced in section IV.F with the description of static and

dynamic ticket frames. The following section formally defines a *reference* as an action specific to a connector-based agent and connector-based multi-agent simulation.

a. References

References are a special type of action that create a connection with a contextually appropriate ticket or action. Using connectors extended in receptor mode to capture the current state (or context) of the agent, the *reference* connects with an action or ticket whose function is described by connectors extended in stimulus mode. Equation 15 defines a reference in terms of its control function and connector set.

$$\textit{Reference } r = \langle f_r, C_r \rangle$$

$$f_r - \textit{control function}$$
$$C_r - r\textit{'s set of connectors}$$

Equation 15. Reference Definition

The function f_r manages the extension and retraction of the connectors from C_r based on input from E_{inner}. In addition, for the extended connectors $C'_r \subseteq C_r$, f_r determines the set of values each of the connectors from C'_r can connect with. For instance, if $c \in C'_r$ is a connector extended in receptor mode, then c is able to establish a connection with another connector whose value is a member of the set $\Phi(c)$. Working in conjunction with connector c's control function, f_r adds or removes values from the set. In this way, the connectors extended by the *reference* are always attempting to connect with actions and tickets that are appropriate to the current state of E_{inner}. Equation 16 defines the conditions under which a *reference* will connect with a ticket. *References* connect to actions in a similar manner as described in the next section (Equation 18).

Given,

$$\textit{Reference } r = \langle f_r, C_r \rangle$$
$$\textit{Ticket } tk = \langle f_{tk}, C_{tk}, F \rangle$$
$$\textit{where,}$$
$$C'_r \subseteq C_r \textit{ is } r\textit{'s set of extended connectors (receptor mode)}$$
$$C'_{tk} \subseteq C_{tk} \textit{ is } tk\textit{'s set of extended connectors (stimulus mode)}$$

then r connects with tk iff:

$$(\forall c_r \in C'_r)\, \exists c_{tk} \in C'_{tk}\, (conn(c_r, c_{tk}))$$

($conn(c_r, c_{tk})$ is defined by Equation 6)

Equation 16. Reference Action to Ticket Connection

b. Actions

Agents interact and modify their inner and outer environments by executing actions. In a connector-based agent, actions are made up of three components (Equation 17); a control function (f_a) used to coordinate the connectors, a set of connectors (C_a), and finally the underlying action that is executed when the action wrapper executes.

$$\boxed{\begin{array}{c} \textit{Action } a = \langle f_a, C_a, act \rangle \\[6pt] f_a - \textit{control function} \\ C_a - \textit{set of connectors } (C_a \subseteq C_i \cup C_e) \\ act - \textit{underlying action} \end{array}}$$

Equation 17. Action Definition

References, as defined above, are also capable of connecting with actions. Equation 18 defines the conditions for the connection.

$$\boxed{\begin{array}{c} \textit{Given,} \\[6pt] \textit{Reference } r = \langle f_r, C_r \rangle \\ \textit{Action } a = \langle f_a, C_a, act \rangle \\ \textit{where,} \\ C'_r \subseteq C_r \textit{ is } r\textit{'s set of extended connectors (receptor mode)} \\ C'_a \subseteq C_a \textit{ is } a\textit{'s set of extended connectors (stimulus mode)} \\[6pt] \textit{then } r \textit{ connects with } a \textit{ iff:} \\[6pt] (\forall c_r \in C'_r)\exists c_a \in C'_a\, (conn(c_r, c_a)) \\ \scriptstyle (conn(c_r,\, c_a) \textit{ is defined by Equation 6}) \end{array}}$$

Equation 18. Reference Action to Action Connection

Actions may require certain resources in order to execute. The resource requirements are captured in the control function. Through a connector's *connect* action; an action can bind to an object to fill its resource needs. Figure 24 shows an example where an *EAT* action connects to a *FOOD* object resulting in a connector extending on the action which allows it to connect to the *reference*.

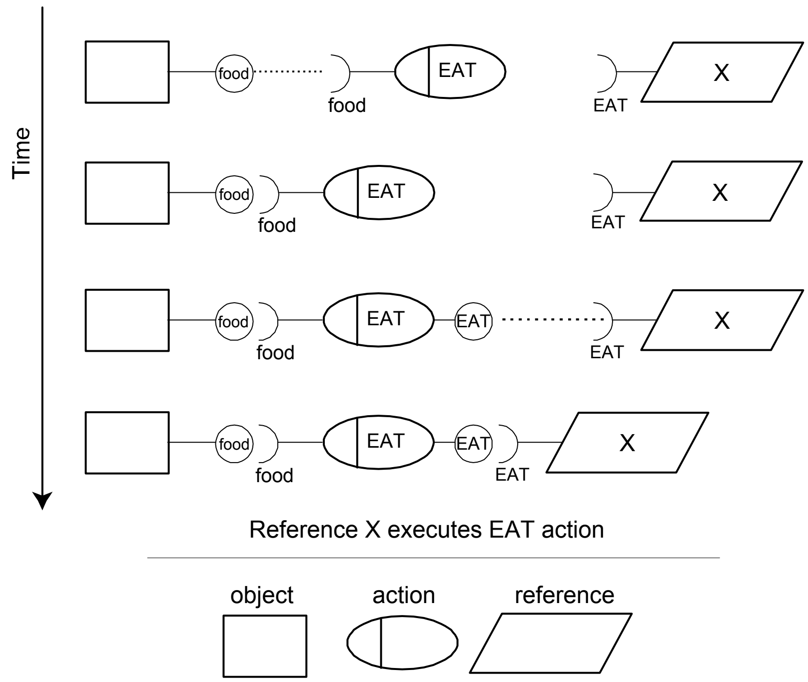


Figure 24. Eat Action Binding to Food Object

c. Signal Cascades

Signal cascades, as described in section IV.E.1 are a sequence of actions and state changes that occur in a cell as the result of external stimuli. When the internal components of a cell are aligned in just the right states, a single stimulus can trigger a complex sequence of internal actions (Figure 13).

Through the combined use of connectors, references, tickets and actions, the power of signal cascading can be simulated in agents. Figure 25 depicts a cascade that is initiated when $agent_i$ connects with $agent_j$. In this example, the external connection triggers an α signal being sent to reference W. W in turn updates the label set on its extended receptor allowing it to connect with ticket P. The first frame of P executes action A; the second executes reference X, resulting in execution of action E. The third frame executes reference Y that binds to ticket Q, resulting in the eventual execution of actions F and B.

If the internal state of $agent_j$ had been different, it is possible that the connection would have triggered a different set of actions, or $agent_j$ may have failed to connect with $agent_i$ all together.

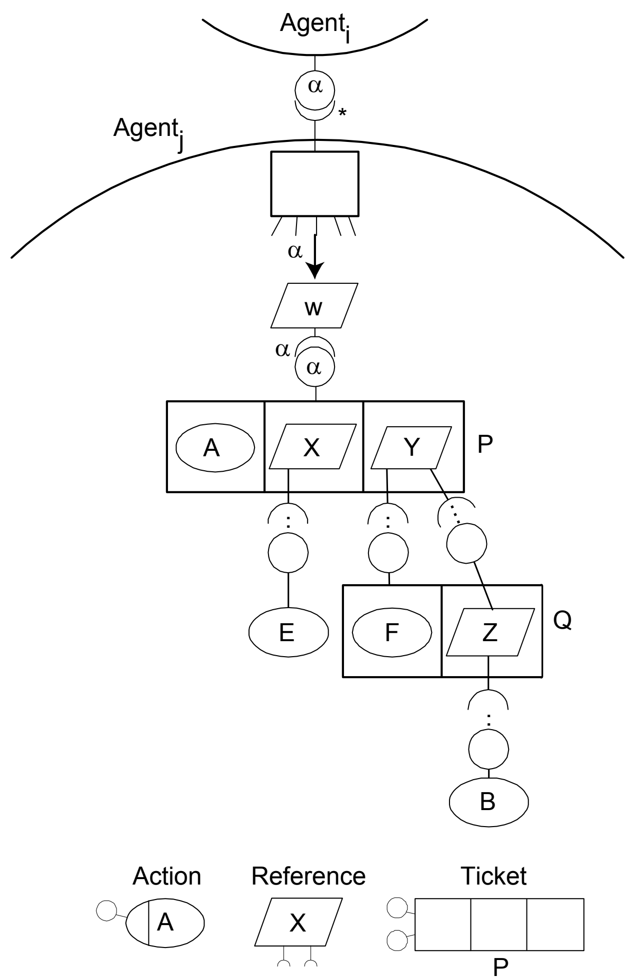


Figure 25. Signal Cascade

3. Agent Connector Sets

An agent's connector sets (C_i and C_e) form the backbone of its intra-agent and inter-agent communication, coordination and control system. Virtually every aspect of the agent is influenced in some way by connectors. The set C_i is defined locally within the agent and is used strictly in an intra-agent role to meet the specific agent's needs. The scope of these connectors does not exist beyond the agent.

Defining the origin of C_e is more difficult. Depending of the developer's point of view, C_e can be defined as a subset of the CMAS's connector set C, ($C_e \subseteq C$). This implies that C is a fixed set that was defined in a top-down fashion and all external connectors must come from this pre-defined set. Alternatively, the CMAS connector set C can be defined as the union of the connector sets from all of the agents and objects in

the system. In this case, C is defined in a bottom-up fashion and could conceivably change over time. Considering the biological inspiration behind connectors, evolutionary theory would lead us to the conclusion that the connectors adapt and evolve locally to meet the needs of the host organism. This is fine in theory, but when it comes to building connector-based software systems, allowing connectors to evolve has not yet been explored.

In the connector-based architecture presented here, it is assumed that C_e is defined at the agent level, but connectors do not evolve, and the CMAS connector set C is the union of the connector sets defined by the objects and agents. In this way, when new agents and objects are introduced to the system, it is possible to include their connectors in the CMAS set C (Equation 27 later in this chapter).

4. Reactive Agents and Goals

The behavior of an agent can be thought of as the manifestation of the actions the agent takes in pursuit of its goals. As described in Chapter IV, the reactive agents (RA) of a Composite Agent function to control the agent's behavior. For any single behavior, there may be multiple goals responsible for producing the behavior. In the Composite Agent architecture, RAs are defined with the intention that a single RA be responsible for a single behavior (Equation 19).

$$a_R = \langle f_R, G_R \rangle$$

$$f_R - \textit{control function}$$
$$G_R - \textit{goal set for } a_R\,;$$

Equation 19. Reactive Agent Definition

With behaviors emerging from multiple goals, a_R is basically a goal management apparatus where f_R manages goal set G_R. G_R is the goal set for a single RA; the Composite Agent's goal set G is defined in Equation 20 as the union of the goal sets for each of the reactive agents.

$$\begin{array}{c}\textit{Given agent a with reactive agent set } A_R, \\ \textit{if } G_k = \textit{goal set for reactive agent } a_{R(k)} \in A_R \\ \textit{then} \\ G = \bigcup_k G_k \text{ ; } k = 1,..,|A_R|\end{array}$$

Equation 20. Agent Goal Set (G)

Equation 3 from Chapter IV defines goals as being comprised of a state, measurement method, weight and set of actions. This definition is modified slightly for connector-based composite agents. For connector-based agents, the "set of actions" is refined to the point where a goal can take one of three possible actions. First, it can execute a "by-name" call to a specific action from the agent's action set. This obviously results in execution of the designated action. Second, a "by-name" call can be made to a specific ticket. In this case, the ticket is executed as described in Chapter IV, eventually resulting in a single action or series of actions being performed. Finally, a reference can be executed which results in the run-time binding and execution of a contextually appropriate ticket or action. References allow the active goal to select an action that not only helps achieve the goal, but is also appropriate for Composite Agent's current state. In other words, it results in context sensitive action selection. When it comes time for the agent to act, the reference from the active goal is evaluated resulting in a connection to one the Composite Agent's tickets or actions (Figure 26).

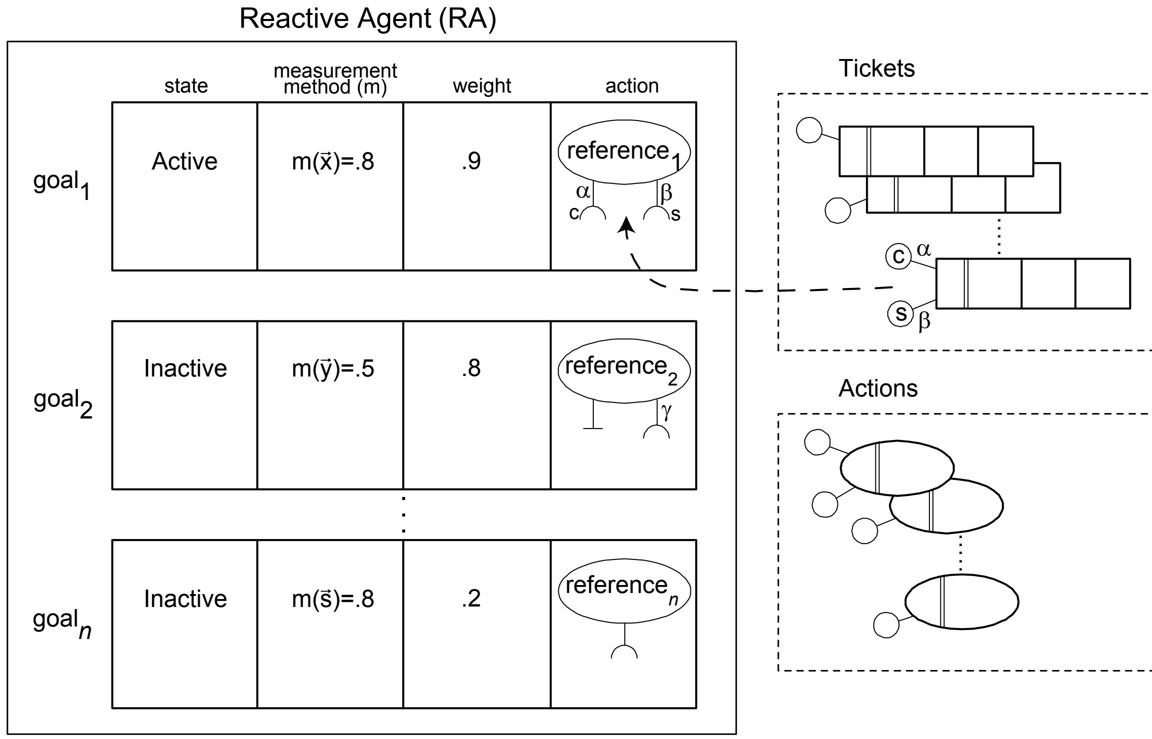


Figure 26. Reactive Agent Goal Structure

The architecture of a Composite Agent is intended to accommodate individual agents with a tremendous range of behaviors. However, even with a single RA and small set of goals, a tremendously complex range of behavior is possible.

F. OPERATIONS

Ferber describes operations as actions that make it possible for the agents to perceive, produce, consume, transform and manipulate the objects and agents in the environment [Ferber, 1999]. There are three actions that are unique to connector-based multi-agent systems, and set CMASs apart from other agent-base architectures. The first is the fundamental connector level action of *connect*. Equation 6 defined the conditions under which two connectors are able to connect. The *reference* action (Equation 15) was constructed from connectors to provide agents with the capability to select and execute contextually appropriate tickets and actions. The final action, called *connecting* or a *connection* action, allows agents to interact with other agents or objects. All three of these CMAS unique actions, *connect*, *reference* and *connecting*, are assumed to be part of every agent's action set. From this description, the following definition for Operations is derived (Equation 21).

Given CMAS M with agent set A,
M's operations set Ω is defined as:

$$\Omega = \bigcup_i \Omega(a_i) \quad i = 1,.., |A|$$

$$a_i \in A$$

$$\Omega(a_i) = \text{action set for agent } a_i$$

Equation 21. CMAS Operations Set (Ω)

1. Connecting

Connecting is the process by which agents become bound to, and interact with, other agents and objects in the environment. By connecting, agents establish a *connection* and are said to be *connected*. During a connection, or while connected, agents interact and communicate with the bound entities. Connections are established and maintained across a set of connectors. The bound connectors provide the communication channels for the connected entities. The entities engaged in the connection have the full range of communication and coordination actions provided by the bound connectors at their disposal. Section IV.E.2 provides a detailed description of connector actions.

Equation 22 describes the conditions under which two agents are able to connect. A connection can be established between agent_i and agent_j if there exists a subset of connectors from agent_i's set of extended connectors (receptor mode) ($C''_{e(i)}$) such that for every $c_{e(i)} \in C''_{e(i)}$ there exists a matching connector on agent_j extended in stimulus mode. Individual connectors match if they meet the criteria established by the predicate $conn(c_i, c_j)$ (Equation 6). Equation 23 is a similar equation that describes the conditions under which agents connect to objects.

Given agents a_i and a_j with external connectors sets $C_{e(i)}$ and $C_{e(j)}$ respectively,

if

$C'_{e(i)} \subseteq C_{e(i)}$ *are extended in receptor mode*
$C'_{e(j)} \subseteq C_{e(j)}$ *are extended in stimulus mode*

then a_i is able to connect with a_j ($conn(a_i, a_j)$) iff:

$$(\exists C''_{e(i)} \subseteq C'_{e(i)})(\forall c_{e(i)} \in C''_{e(i)})\exists c_{e(j)} \in C'_{e(j)}\ (conn(c_{e(i)}, c_{e(j)}))$$

($conn(c_{e(i)}, c_{e(j)})$ is defined by Equation 6)

Equation 22. Agent-to-Agent Connection Conditions – $conn(a_i, a_j)$

Given agent a_i with external connector set $C_{e(i)}$ and object o_j with connector set C_j,

if,

$$C'_{e(i)} \subseteq C_{e(i)} \text{ are extended in receptor mode}$$
$$C'_j \subseteq C_j \text{ are extended in stimulus mode}$$

then a_i is able to connect with o_j ($conn(a_i, o_j)$) iff:

$$(\exists C''_{e(i)} \subseteq C'_{e(i)})(\forall c_{e(i)} \in C''_{e(i)})\exists c_j \in C'_j\ (conn(c_{e(i)}, c_j))$$

($conn(c_{e(i)}, c_j)$ is defined by Equation 6)

Equation 23. Agent-to-Object Connection Conditions – $conn(a_i, o_j)$

Equation 22 and Equation 23 establish the criteria under which it is possible to connect. The actual decision to initiate a connection comes from one of two sources. Connections are initiated internally by an agent or externally by a higher-level control mechanism in the simulation. In [VanPutte, 2002], an external mechanism called an *ibinder* is used to bring together agents and objects with matching *iconnectors*. Agents and infrastructure objects register with the *ibinder*. When resources or vulnerabilities are exposed via connectors by the infrastructure objects, the *ibinder* is notified. Likewise, the *ibinder* is aware of the resource requirements of the agents. The *ibinder* serves as a digital switchboard, connecting agents to resources.

The story engine, which is described in the next chapter, strictly uses agent-initiated connections to generate stories. Agent-initiated binding is a self-serving process where agents look for connections that are advantageous to themselves and promote their current goals. This does not mean the agents cannot or will not cooperate. On the contrary, connections can be weighted so that mutually beneficial connections are more likely to occur.

The agent has the following information available when evaluating the fitness of a connection: its internal state, its goals with their status and measure, and the type and value of the connectors it is evaluating for connection. In addition, *$conn(a_i, a_j)$*[3] being true guarantees that the connection is valid with respect to the laws of the story world. From the set of possible connections the agent can engage in, it must determine which one appears to be the most beneficial based on its goals and current state.

[3] The *conn* predicate provides a means of formally describing the conditions under which a connection may occur. It is not meant to imply that connector-based composite agents, or any connector-based agent must use formal logic methods to manage their connection process.

At any given time, an agent is aware of a set of entities (objects and agents) in its environment. This set of entities W, is described as the agent's *awareness set* (Equation 24).

$$\begin{gathered}\textit{Given}\\ \textit{CMAS } M \textit{ with agent set } A, \textit{ object set } O,\\ \textit{and}\\ \textit{agent } a \in A,\\ \exists W \subseteq A \cup O,\\ \textit{s.t. } a \textit{ is aware of the agents and objects of } W.\\ \textit{Agent } a \textit{ maintains } W \textit{ through its extended connectors and SCAs.}\end{gathered}$$

Equation 24. Awareness Set (W)

Given a with awareness set W, there is a set $W' \subseteq W$ that a is able to connect with (Equation 25). W' is known as the agent's set of *candidate connections*. Note that it is possible for $W' = \varnothing$ in which case the agent cannot engage in a connection.

$$\begin{gathered}\textit{Given agent } a \textit{ with awareness set } W,\\ a\textit{'s set of candidate connections is defined as:}\\ W' = \bigcup_{e \in W} conn(a, e)\end{gathered}$$

Equation 25. Candidate Connection Set (W′)

Equation 26 defines an evaluation function for comparing the respective value of each connection when multiple connections are possible.

$$\begin{gathered}\textit{Given agent } a \textit{ with external connector set } C,\\ \textit{goal set } G, \textit{ and candidate connections } W'.\\ (\forall e \in W')\ \exists C' \subseteq C \textit{ for which } conn(a, e) \textit{ is true.}\\ C' \textit{ is the set of connector(s) over which } a \textit{ can connect to } e.\\ \textit{The most favorable connection (mfc) over the set } W' \textit{ is}\\ mfc = \max_i f(C'_i, G, \xi(a));\ i = 1,..,|W'|\\ \xi(a) \textit{ is an evaluation of agent } a\textit{'s state.}\\ f \textit{ is a domain specific evaluation of the value of connection } i.\end{gathered}$$

Equation 26. Most Favorable Connection (mfc)

G. RELATIONS

In a previous chapter, relations were described as "abstract links" that create a dependency between the agents. Connectors provide a natural means for establishing and

maintaining relationships between agents. They can be used to establish private networks between agents. The "related" agents can then communicate and coordinate over the private network.

An agent's set of extended connectors defines its scope of interest and influence. By their type and value attributes, they define the other agents and objects in the environment the host agent is interested in interacting with. This scope of interest and influence is precisely what is at the heart of relationships. The "abstract links" used to describe relations have a one-to-one correspondence with connector types. When a group of agents shares a set of common connectors, they have the ability to interact as a coordinated group over a "private" communications network. The level and type of coordination is determined by the "relationship-specific" actions the related agents possess.

The definition of connectors allows for the basic action set of a connector to be augmented with "type-specific" actions. When a set of relationship-specific connectors is defined, these type-specific actions are used to implement actions that are unique to the relationship.

Figure 27 shows a CMAS environment populated with agents and resources. There are three types of resources (r, s, and t). Each agent has a primary goal of collecting as many resources as possible, but only of a single type. Connector type β is used to sense and bind to resource objects. Agent D, for instance, can sense and collect resource r. The agents have a limited perception range and randomly explore the environment in search of resources. Agents A, B, and C have established a cooperative relationship whereby they share information and notify each other when they locate a resource of interest to one of the other agents. The relationship is established through a type λ connector. By entering into this relationship, the agents are able to sense all three resources, not just their primary resource. When an agent detects a resource that is of interest to another agent in the relationship, it extends a type λ connector in stimulus mode. The interested agent establishes a connection through which the resource type and location are passed. This relationship greatly extends the related agents' sensory range.

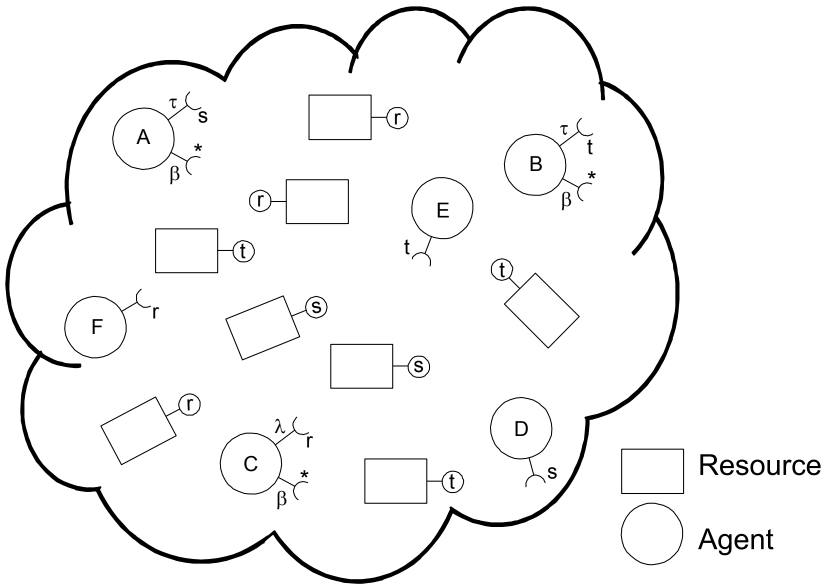


Figure 27. Resource Sharing Relationship

While this example is relatively small, it serves to demonstrate the role connectors play in establishing and defining relationships in a CMAS simulation.

H. LAWS

The laws of the CMAS are both global and distributed. They are intended to establish the boundaries for the simulation. They do not define specific paths or behaviors; those are left for the agents to discover. In an earlier chapter, the ideas of semi-fluid software structures and *indirect* solutions were discussed. Agents are useful for finding solutions the designers may not have considered. To do this, they must have the freedom to operate across a broad landscape. However, if the solution is going to be useful, there must be constraints on the landscape. These constraints are the *laws* of the simulation. They must be chosen carefully so as not to overly constrain the agents, yet be specific enough so the solutions are viable. This section does not attempt to define specific constraints for they are very much domain dependent; that is saved for the following chapters where the story engine and *America's Army: Soldiers* are described. The remainder of this section identifies the components used to define and enforce constraints.

In agent-based modeling, there is no single omniscient entity controlling the actions and interactions of the agents. With no central control, the enforcement of laws is

pushed down to the agent level. This is not to say that there are no global laws. Newton's laws are a good example; they may apply to all agents and objects in a situated environment involving motion. One set of equations can be used to describe the laws, but there is no centralized control mechanism that is responsible for ensuring each agent and object adheres to these laws. They are enforced by their encoding in the actions of the agents, i.e., if an agent at point x moves with a velocity vector of v for t seconds, it will end up at point y.

In the CMAS architecture, laws are also described and enforced through *connect* and *connection* actions, and in terms of *tickets* and *references*. All of these work together to define and enforce the laws.

Chapter VII describes an implementation of the story engine set in the domain of the U.S. Army. The engine generates stories depicting a soldier's progression through an Army career. In this simulation, tickets and references are used to capture the procedural constraints, or laws, associated with Army career progression and school requirements to achieve specific designations. The tickets ensure the soldier follows a legitimate Army career path. For example, when soldiers enlist in the Army, there are specific training paths based on their specialty. These include *entry processing*, *basic training*, *advanced training* and then a duty assignment. This progression is captured in tickets that ensure a soldier completes basic training before advanced training. It would not make sense for the soldier to jump directly to a duty assignment with no training.

Within each training area, there are certain events every soldier must participate in as they progress. These progressions are not optional; they are specific *laws* that apply to the Army. Army basic training is made up of a well-defined sequence of training events that prepare a soldier for the next phase of training. The events are not optional, they must be completed, and usually in a fixed sequence. This sort of sequencing is captured in tickets.

In conjunction with tickets, connectors and references are used to enforce prerequisite constraints. Continuing with the Army career progression example, at some point a soldier must select their next duty assignment. The action to select occurs when the soldier reaches the *frame* of their duty assignment *ticket* marking the end of the

assignment. The *reference* in the ticket frame is populated with connectors that represent the soldier's skill set, experience and performance. The only schools and duty assignments the soldier can select are those for which a valid connection is possible. If the soldier wants to join the Special Forces but does not possess the correct connector set, they will not be able to connect with the Special Forces ticket. In this way, tickets, references and connectors work together to enforce the laws of the simulation. The laws pertaining to qualifying for Special Forces are captured in the set of connectors extended (stimulus mode) from the Special Forces ticket.

I. CONNECTOR SET

The set of connectors comprising a CMAS can be separated into two categories. There are external connectors that are visible throughout the CMAS environment *E*. There is also a set of internal connectors that the agents use for intra-agent coordination and control. Equation 27 defines the two sets as the union of the corresponding set from each of the system's agents and objects.

Given,

CMAS M with agent set A and object set O,

$$a_j = \langle A_{SC(j)}, T_j, \Omega_j, E_{inner(j)}, C_{i(j)}, C_{e(j)}, G_j, A_{R(j)} \rangle;\ (a_j \in A)$$

and

$$o_k = \langle f_c, C_{o(k)} \rangle\ (o_k \in O)$$

then,

$$C = \langle C_{int}, C_{ext} \rangle$$

where,

$$C_{int} = \bigcup_j C_{i(j)}\ ;\ j = 1,..,|A|$$

$$C_{ext} = \left(\bigcup_j C_{e(j)} \right) \bigcup \left(\bigcup_k C_{o(k)} \right);\ j = 1,..,|A|,\ k = 1,..,|O|$$

Equation 27. CMAS Connector Set (C)

J. MANAGING THE COMPLEXITY OF AGENT INTERACTIONS

Connectors serve to establish a context for the agents within their environment. Context is defined as "the interrelated conditions in which something exists." [Merriam-Webster, 2002]. The context influences the agent's goals and actions. For example, when a person is sitting in a meeting, their surroundings and the meeting itself establish a

context for the person that helps to determine what actions are appropriate and which are not. It also influences the person's goals.

An agent's context is established by its connectors. Therefore, given the agent's set of extended connectors, only certain tickets and actions are possible, either directly or through *references*. As soon as the agent's connectors change state, the context immediately changes and new actions and tickets are available. Agents may have extensive connector sets that allow for huge number of states (or contexts). The soldier characters in the *America's Army: Soldiers* have seven core values each with five levels, six resources, each with five levels, five possible active goals with each goal being in one of five states, and they can be in any one of four career phases and any one of 10 types of places. From this, a soldier agent's context may be any of 1.6×10^{15} contexts. However, the important notion is that at any time, the soldier only has a single context. The agent's context is captured by its extended connectors, which are managed locally by the agent. In managing connectors (context) locally, the agent self-regulates its interests and the factors that influence its actions. Connectors are the common thread that tie the state of the agent's outer environment together with the inner environment, the agent's goals, and through *references*, the appropriate tickets and actions to achieve the goals.

K. CONNECTOR-BASED SIMULATION MODEL

The connection action is at the heart of CMAS simulation. When a CMAS environment, including all of its agents and objects, is described in terms of connectors, the connecting process provides a means of managing the combinatorial explosion of possible states and contexts, to facilitate agent and object interaction. In the course of the simulation, the agents act on two levels, the first being the "reactive" response based on connector input. Stimuli received through connectors trigger signaling cascades within the agent. The second is the action(s) the agent takes in pursuit of its goals. This action is the product of the agent's goal management process. The CMAS simulation model brings these two together.

```
While (not done) {
   Randomize agents
   For each agent {
      Sense environment
         SCAs and connectors sense environment
         Update existing bound connections (push and pull)
         Update E_inner
      Connect
         Evaluate mfc function and initiate new connections
      Update new connections
         While (new connections exist) {
            Update new connections (push and pull)
            Update E_inner
            Evaluate mfc function and initiate new connections
         }
      Act
   }
} // end main loop
```

Figure 28. CMAS Simulation Model

L. SUMMARY

This chapter presented a formal definition of the Connector-based Multi-Agent System (CMAS) and described a simulation model based on the CMAS architecture. The story engine presented in the next chapter is constructed from the CMAS architecture.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. INTERACTIVE STORY GENERATION SYSTEM

A. INTRODUCTION

The Interactive Story Generation System (ISGS) is comprised of three primary components that work in concert to structure and present interactive stories (Figure 29). These components are the story engine, scene rendering subsystem (SRS), and graphical interface. The ISGS follows the Model-View-Controller (MVC) architecture [Gamma *et al*., 1995]. MVC divides the system responsibilities into three parts: the *model*, which contains the program data; the *view*, which provides the visual presentation of the model, and the *controller*, which defines the system behavior. The story engine CMAS and associated data represent the controller and model, while the SRS and graphical interface provide the view. The graphical interface augments the controller by way of manual control in the form of user intervention and interaction with the stories. Following the MVC abstraction, replacing the SRS with another *view* can be performed without need to modify the story engine.

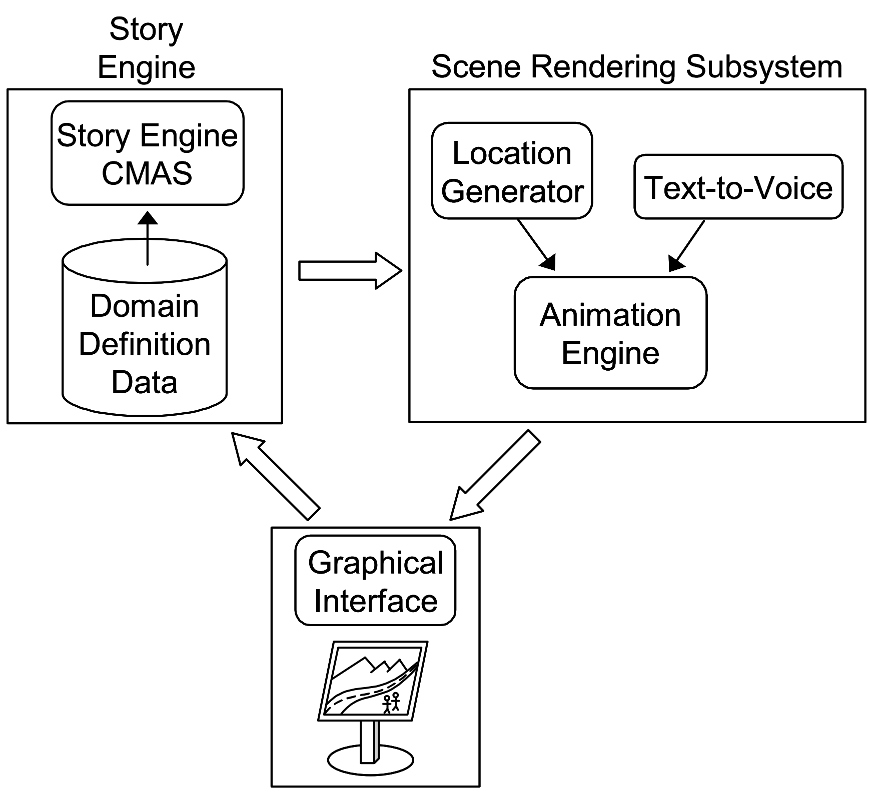


Figure 29. Interactive Story Generation System

This dissertation focuses on the story engine and story engine CMAS. First, this chapter relates story worlds and story lines to the CMAS architecture. The components of the story engine CMAS that uniquely set it apart from other CMASs are then presented. An inductive argument is presented as proof that the story engine generates story lines that are logically connected and goal-directed. Further, it is argued, that the story lines are sound with respect to the domain being modeled. Finally, the SRS and graphical interface are described as components of the integrated ISGS. The ISGS brings together the story engine CMAS with generative text-to-voice, photo-cell animation, and photo-realistic settings (locations) to present interactive stories.

B. STORY ONTOLOGY

This section describes the narrative constructs of story world, story and story line as they apply to this work. A one-to-one mapping is defined between these narrative structures and the corresponding CMAS constructs of the story engine.

1. Story World, Story and Story Line

A *story world* is comprised of characters and props, along with locations where events occur and characters interact with each other, the props, and their surroundings. The interactions are not random nor are the characters free to act as they wish without regard to the rest of the story world. There are constraints on character actions and interactions. Equation 28 defines a *story world* in terms of five components: characters, props, locations, actions and constraints.

$$\boxed{\begin{array}{c} \textit{Story World} = \{Ch, P, L, A, Co\} \\[4pt] Ch - \textit{Characters} \\ P - \textit{Props} \\ L - \textit{Locations} \\ A - \textit{Actions} \\ Co - \textit{Constraints} \end{array}}$$

Equation 28. Story World Definition

Consider the children's story of *The Three Little Pigs*. In the *story world* of the three pigs, the world consists of three pigs, a wolf, hay, sticks, bricks, plus additional props (flute, fiddle, shovels, etc.) (Figure 30). The action takes place in multiple locations including a straw house, wood house and brick house. There are constraints on what the characters can do, though in this case they are very much controlled by the

author as opposed to constrained by physics and nature. Multiple story lines based on this story world are possible, but actually generating a *story line* requires that a set of *initial conditions* be set. In this case, the *initial conditions* might include the personality traits for the pigs and wolf, along with their skill levels and initial goal(s). Therefore, a traversal requires a *story world* plus a set of *initial conditions*. This tuple of (*story world*, *initial conditions*) defines a *story*.

Continuing with the three little pigs, if the initial conditions define two happy-go-lucky pigs, one hard working pig and a hungry wolf, the result may be similar to the traditional story line of the three pigs. On the other hand, by modifying the initial conditions and defining three devious pigs with a good-natured wolf, it may be possible to get a completely different story line from the same story world, i.e., *The True Story of the 3 Little Pigs* [Scieszka and Smith, 1989]. A *story line* is therefore the result of a traversal through a given *story* (*story world* plus *initial conditions*).

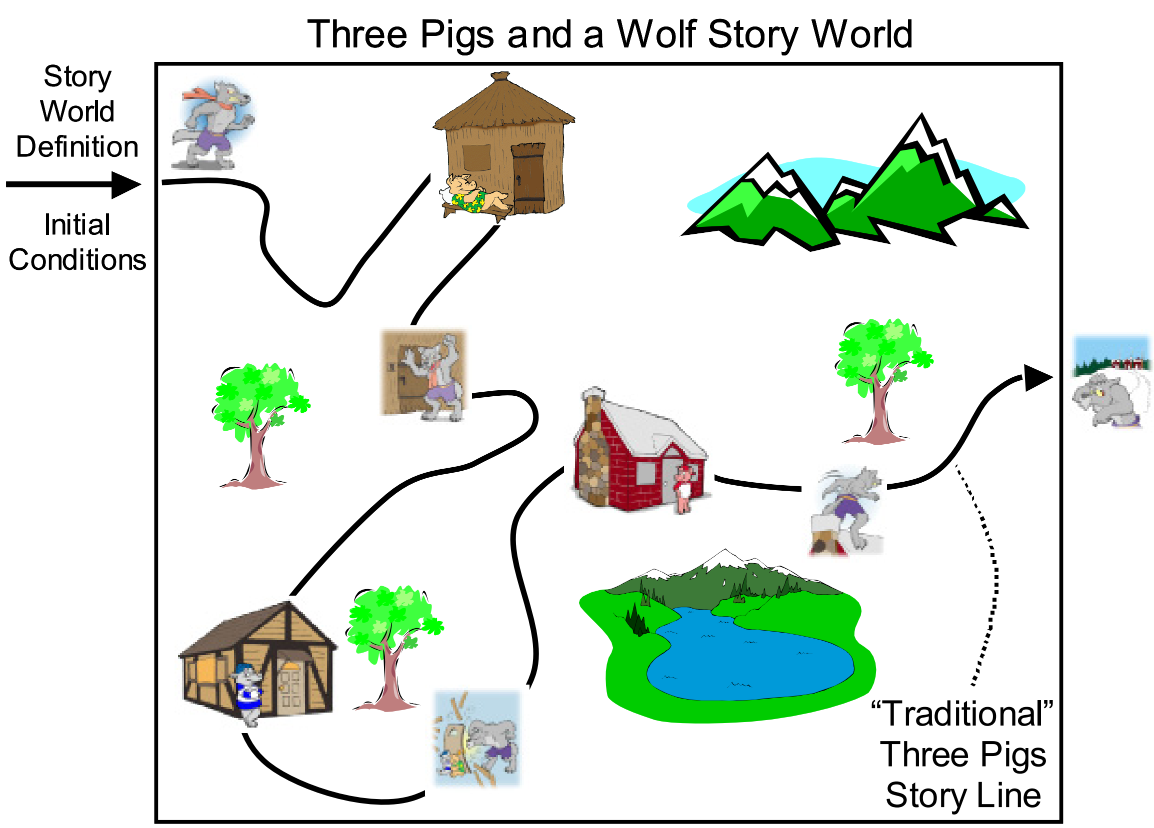


Figure 30. Three Pigs and a Wolf Story World

The definition of the *story world* provides the "potential" for numerous and varying *stories*. When the *story world* is transferred to software that "potential" can be realized in the form of dynamic *story lines*.

2. Story World as a CMAS

The *story world* correlates directly with the definition of a CMAS (Equation 8). The CMAS components of agents, objects, and environment equate to the *story world's* characters, props and locations respectively. The laws, operations and relations of the CMAS establish the constraints within which the characters interact. The goals provide the impetus for the characters while the tickets and actions provide the means for the characters to achieve their goals. When the CMAS is combined with a set of initial conditions, the result is a simulation.

In this context, a simulation is a generating function that produces a *story line.* Let *S* be defined as a *generating function* that when applied to a CMAS produces a set of outputs, including a story line (Equation 29).

Given
CMAS M and initial conditions I, *S is generating function that produces output as follows:*
$S(M,I) = O;$
where *O is a set of outputs and story line* $L \in O$*.*

Equation 29. CMAS Generating Function (S)

S takes initial conditions and a CMAS as input, and produces a set of outputs, including a *story line*, *L*. *L* is called a *story line* generated by *S* given initial conditions *I*. When *S* is left to run to completion without intervention, *S* is said to be *non-interactive*. If allowances are made for the user to intervene as the simulation progresses, as is the case with the story engine, then *S* is *interactive*.

Repeated applications of *S* may generate identical or different sequences *L* each time *S* is applied with the same initial conditions. When repeated application of *S* generates identical story lines, *S* is called *deterministic*. Alternatively, when *S* generates a different story line each time it is applied, the simulation is said to be *non-deterministic*.

When S is *interactive*, it is likewise *non-deterministic* as a consequence of the variability in user intervention.

Unlike most stories produced by systems with statically defined plans, story engine CMAS models are purposely designed to produce novel or surprising story lines, i.e. non-deterministic story lines. There are two ways to do this: (1) by introducing pseudo-randomness in the agent's plans through Monte Carlo methods, or (2) by permitting the agents to devise their own plans as the story line unfolds. This dissertation distinguishes between Monte Carlo techniques and dynamic plans, e.g. plans that are undefined initially, and unfold dynamically as the simulation runs.

The story engine is not concerned with finding a single plan or story line; it is intended to generate "possible" story lines. But rather than produce "possible" story lines through the application of Monte Carlo techniques, the story engine generates story lines by allowing agents to dynamically construct plans as they explore a story world, (as defined by the story engine CMAS data set) in pursuit of their goal(s). The resulting story lines are plans that lead to the achievement of a goal, and while achievement is significant, the path to achievement is also of interest.

This agent-based approach to generating story lines does not exclude Monte Carlo techniques from being used to generate multiple replications in order to study the CMAS model of the domain. Examination of the frequency of possible story lines can give insight into the CMAS representation and also the likelihood of the answers occurring based on the agent's representation of the domain being studied.

The space of potential story lines is combinatorially large and therefore it is not possible to explore the entire space and visit every possible story line. The best that can be done is to conduct a focused exploration. In the previous chapter, it was shown how the focus of the agent is controlled via connectors. In essence, the connectors tune the agent's attention to precisely those parts of the environment (agents and objects) most relevant given the agent's current state; and allow it to communicate and coordinate with precisely those agents that can assist it in achieving its goals. This modeling approach focuses on the middle ground between finding a single answer and exploring all possible answers.

C. STORY ENGINE CMAS

The story engine CMAS is a data-driven architecture. The simulation domain is defined by the data, not the CMAS. In the next chapter, a data set representative of U.S. Army career progression is described. This data is used in *America's Army: Soldiers* to generate stories centered on pursuing an army career. By changing data, the story engine CMAS can be used to generate stories within a totally different domain.

The story engine, as depicted in Figure 29, is a simulation kernel that implements the generating function S defined by Equation 29. The story engine is designed to operate on a specific instance of a CMAS, called a story engine CMAS. The story engine is a non-deterministic, interactive engine that explores a story world, as defined by a story engine CMAS and associated data set, to generate story lines. The non-deterministic characteristic of the story engine results from user intervention and the possible use of pseudo-random numbers to make choices as the story line is generated. The interactive nature of the engine does not impact the repeatability of the story lines. If interaction by the user is considered part of the input, albeit occurring at varying points in the process, then interactively generated story lines are repeatable given identical initial conditions, random number seeds, and the same user interaction at the same times and places in the story line generation process.

The story engine CMAS follows the general constructs of a CMAS as described in Chapter V. The unique components are two connector-based composite agent types; *character agents* and *scene agents*. The character and scene agents are capable of generating dynamic stories through a repetitive connection process. These two agent types are combined with tickets, actions, references, connectors, laws and relations to define a story world. Through the ISGS, the story world can be combined with imagery and sound to generate and visualize multiple story lines (Figure 31).

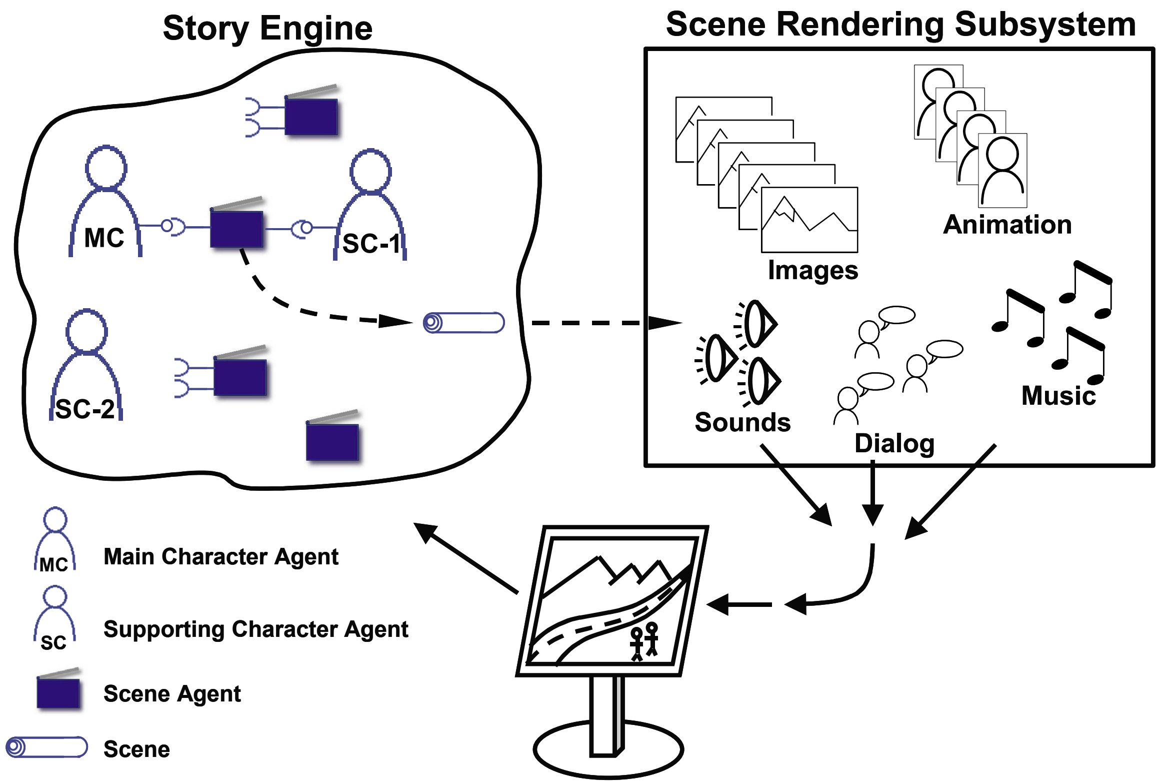


Figure 31. Interactive Story Generation System with Agents and Media

The remainder of this section describes character agents, scene agents and the character-to-scene connection process. The following vignette is repeatedly referenced as the story engine CMAS components are described.

> A young man wants to buy a car, and on a Saturday afternoon he heads to the local used car dealership. The experienced used car dealer greets his customer and quickly sizes him up as a naïve buyer. As they begin walking through the car lot, he assures the young man that he has the perfect car for him. The scene closes with the young man driving away in the "perfect car" having just signed a high interest loan, while paying well over the car's value.

1. Character Agents

The story engine CMAS structures stories around a central character agent. This central character is designated the "main character" and is the protagonist of the story. The implication of this, from the CMAS perspective, is that connections with scene agents are always initiated by the main character. In the ISGS, the player's character is

the main character. In this way, the story is structured around the player's interventions and their character's goals.

Character agents are composite agents, and as such, they are able to take full advantage of the capabilities provided by the connector-based composite agent architecture. The character agents described here closely reflect the agents developed for the initial version of the story engine CMAS constructed for the *America's Army: Soldiers* project. They do not necessarily implement all of the connector-based composite agent features. For example, the character agents do not make use of Symbolic Constructor Agents; their external sensing occurs exclusively through extended connectors. This is not a limitation on the agents; it is simply how the initial version of the story engine CMAS was implemented.

a. Goals and Actions

The agent's inner environment consists of a set of state variables with associated external connectors. The agents are equipped with a set of goals and a reactive agent to manage the goals. The agent has a set of *tickets* it uses to achieve its goals. As described in Sections IV.F and V.E.4, an agent's goals resolve to specific *actions* through *references* and *tickets* (Figure 26). In the story engine, character agents have a limited set of actions that are primarily used to initiate the agent's next connection. Just as actors don't act until they are filling a role in a scene on stage (or in a movie), character agents can't act until they are filling a role in a scene (connected with a scene agent). Because the characters are autonomous agents, they need control over which scenes they participate in. The character agent actions provide this control. An action's precise function is to initiate a connection with a scene agent based on the connectors defined by the action. The *actions* that literally change the character's state, further its goals and propel it through the story are resident in the scene agents.

Once a connection is established, the character fills a role in the scene and is then free to act within the bounds of the scene based on its current state (as expressed through its connectors) and goals. For instance, if the character has a goal of buying a car, then it must connect with a scene that will put it in a position where it can purchase a car. The kind of car purchased, and how good a deal the character negotiates is influenced by how badly the character wants a car (goal weight) and how savvy the character is as a buyer

(state variable). The character knows *what* it wants to do (buy a car); connecting to a scene agent provides the *how* (car lot scene). A detailed description of the character-to-scene connection is provided in a following section.

b. Most Favorable Connection

When a character agent prepares to initiate a connection, there are potentially many scenes it can connect with, but it can only connect with one scene at a time. That is, a character can only be in one place, doing one thing at a time. The character agent initiates a connection with a scene agent based on the evaluation of its most favorable connection (*mfc*) function. As described earlier, goals have a weight and a measurement method. The measurement method provides a numeric measure of how well the goal is being satisfied (higher being more satisfied), while the weight is a measure of the importance of the goal. These values are used by the *mfc* to identify the connection that is "best" in terms of the agent's goals.

Given agent *a*, with goal set *G*, Equation 25 defines a set W' as the set of entities in the environment with which *a* is able to establish a connection. With the story engine CMAS, character agents only connect with scene agents so W' consists entirely of scene agents. Equation 30 defines a character agent's most favorable connection function. The *mfc* is a measure of the goal's weight (wt) times its level of satisfaction, where satisfaction level is 1.0 minus the goal's measure (mm). The logic being that the more satisfied the goal, the less attention it needs at the current time. This is scaled by the goal's importance.

Given agent a with goal set G and
possible connections W′ as defined by Equation 26,

$$mfc = \max_{\substack{g_i \in G \\ s.t.\ conn(g_i,\, W')}} (1.0 - mm_i)(wt_i)$$

$$mm_i \text{ - } g_i\text{'s measure; } (0.0 \leq mm_i \leq 1.0)$$
$$wt_i \text{ - } g_i\text{'s weight; } (0.0 \leq wt_i \leq 1.0)$$
$$(\exists e \in W')conn(a_i, e) \rightarrow conn(g_i, W'),$$
where a_i *is the action bound to* g_i.

Equation 30. Character Agent Most Favorable Connection Function

Figure 32 shows a character agent's goal structure. Each of the goals is bound to an action via a reference – some directly, others through tickets. The actions are

bound to the references across a set of connectors (Equation 18, Figure 26). The bound connectors establish the set of connectors used by the *mfc* function to evaluate the possible connections. When the *mfc* is evaluated, the selected goal executes the bound action, which in turn initiates a connection with a scene agent based on the connectors identified by the action. In this example, a connection satisfying $goal_1$'s connection criteria (connectors α and β) has a value of (.8)(.9) = 0.72. $Goal_2$ and $goal_3$'s connections have a value of 0.56 and 0.247 respectively. Assuming all three connections are possible, then $goal_1$'s connectors (α and β) would be used, along with the character agent's other extended connectors, to connect with a scene agent.

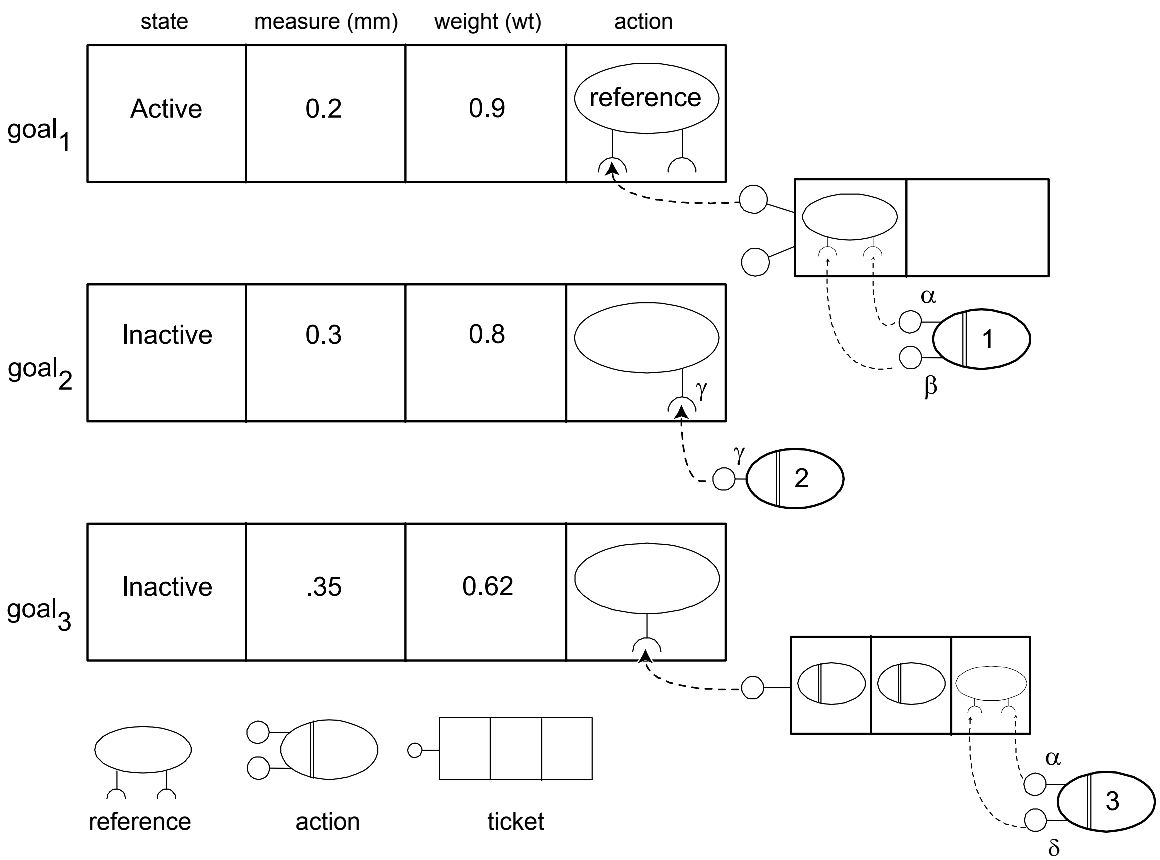


Figure 32. Character Agent Goals with Bound Actions

2. Scene Agents

Scene agents are connector-based composite agents with components and behaviors appropriate for generating scenes of a story. When a character agent connects with a scene agent, a sequence of events is initiated in the agent that results in the

generation of a scene. Just as a story world provides the potential for many story lines, a scene agent provides the potential for many scenes. The character agents connected with the scene agent interact within the bounds established by the scene agent's tickets, connectors (internal and external), and actions in order to generate a scene. This section describes the components and behaviors that are unique to a scene agent (Figure 33).

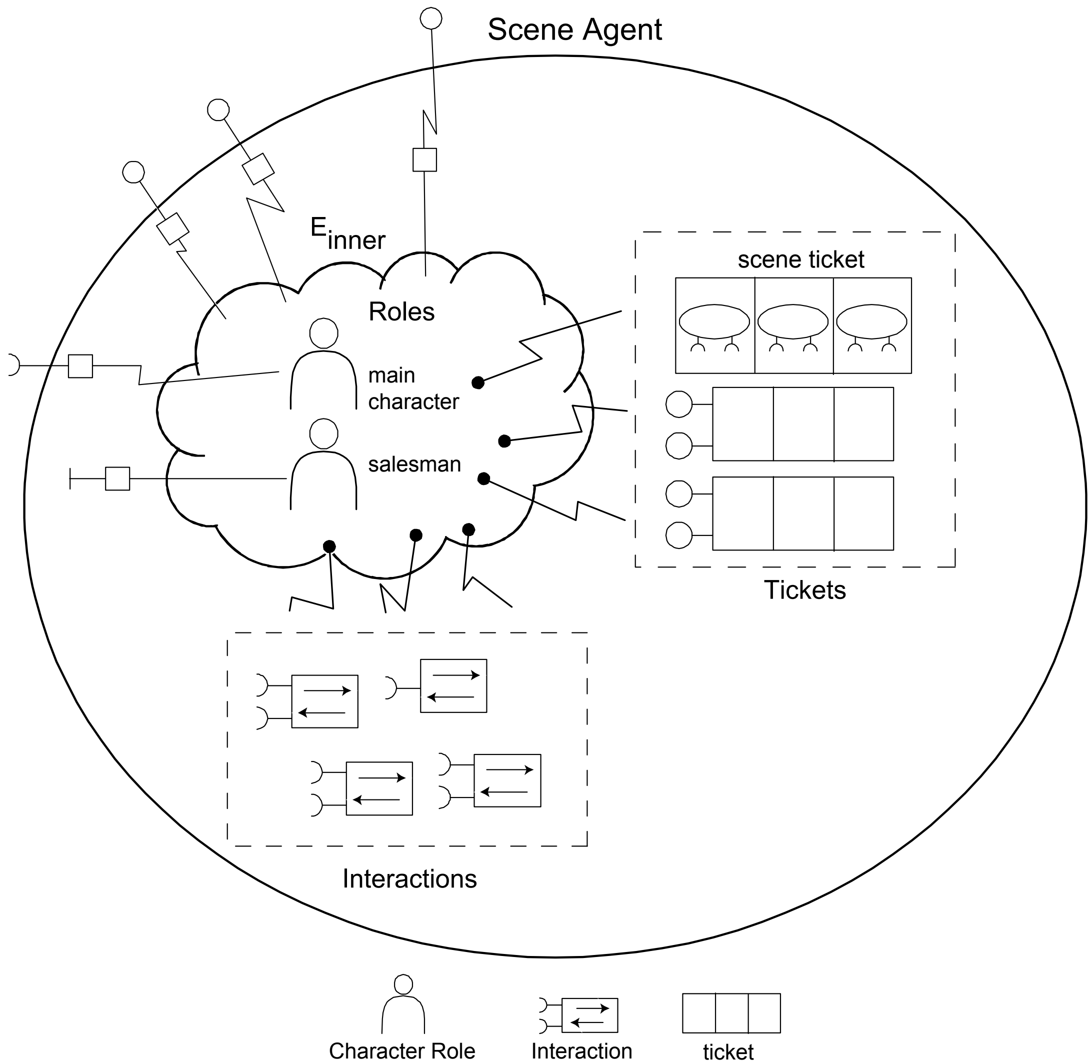


Figure 33. Scene Agent

a. Scenes

The story engine differentiates between a scene agent and a *scene*. When a character agent connects with a scene agent, an internal cascade is triggered within the scene agent that results in the generation of one element of the story line. This element of the story line is described as a *scene*. The *scene* is composed of two parts: story engine

actions and scene rendering actions. In the MVC architecture of the ISGS, these two parts partition the scene into model/controller actions (story engine actions) and view actions (scene rendering actions). The story engine actions are related directly to the story engine CMAS and updating the agents' states. These actions are executed during the connection as the scene is being generated. The rendering actions, on the other hand, are not executed immediately; rather they are captured in a *playlist* to be executed when the connection is complete. The *playlist* is the interface between the story engine and the scene rendering subsystem. The scene, as displayed by the SRS, is an after-the-fact visualization of the actions that occurred during the character agent to scene agent connection.

b. Inner Environment

The scene agent has a set of state variables that describe the "roles" that must be filled for the scene. The roles are filled by character agents. In keeping with the story engine CMAS's character-centric view of a story, one of these roles is always the main character role. The character initiating the connection is automatically cast in the main character role. The remaining list of roles is domain dependent. For example, in *America's Army: Soldiers* the roles include buddy, drill instructor, car salesman, etc. The roles come into play as the scene unfolds. Just as with a movie or play, the role the actor is cast into constrains what the actor can do. The constraints imposed by the story engine are much less restrictive than that of a linear movie or play. In traditional media, there is a fixed script that accompanies the role. The actor's freedom is limited to their range of talents in delivering their lines. With interactive stories, the script is not fixed and the character agent's goals and state play a critical role in determining the outcome of the scene.

c. Tickets

The agent uses tickets to assist in establishing a basic structure for the scene. One of the tickets is designated as the "scene ticket" and is used as a starting point for generating the scene. Figure 33 shows a scene agent with a three-frame scene ticket where each frame contains a reference (Equation 15). In this example, the ticket is logically broken into a beginning, middle and end corresponding to the setup of the scene, the main action and the conclusion. The references in each of the frames connect

to tickets or actions that move the action forward at run-time. This process is described later in this chapter in the Character Agent to Scene Agent Connections section.

d. Connectors: External and Internal

Scene agents use external connectors to sense the outer environment and maintain an up-to-date view of its inner environment. The scene agent's external connector set is partitioned into two subsets. The first is the set of connectors maintained by the scene agent. The second is a set of connectors associated with each of the "roles" in the scene agent. The state and value of these "role" connectors are not established until the character agents connect and fill the roles. This late binding of connectors at run-time allows for a tremendously flexible and contextually sensitive connection process. A special type of action, called an *interaction* is specifically designed to take advantage of this delayed binding. The interactions, with their non-instantiated connectors, create a potential for action. However, that potential does not take shape until the character agents bind to a role.

In addition, scene agents make extensive use of internal connectors to enable the signaling cascade that generates a scene. The internal connectors are a critical component of the agent's internal control mechanism, used to manage its tickets and interactions.

e. Interactions

The car sales vignette described at the beginning of this section could have played out many different ways depending on any number of factors, including the car dealer's personality and integrity, the buyer's experience level, and how well the buyer liked the cars he was offered. Interactions are the components that enable this sort of variation; they use connectors (internal and external) as their primary control mechanism. When a scene is generated during a connection, it is generated on an interaction-by-interaction basis.

Interactions are *compound actions* that bind together at run-time using internal connectors to generate complex and contextually appropriate sequences of events. They are comprised of two types of *atomic actions*: story engine CMAS actions and scene rendering actions. A strict partitioning is enforced to maintain the integrity of the story engine CMAS as a stand-alone controller system in the ISGS MVC architecture.

(1) Story Engine CMAS Actions. Table 1 lists the set of atomic story engine CMAS actions. They are broken into three categories: actions to update state variables (string and numeric), actions to control internal connectors, and an action to manage objects.

	Action Name	Parameters	Description
State Variables	SetStringVariable	role state variable new value	Set the state variable for the character agent filling the designated role to the string value specified by new value.
	SetNumericVariable	role state variable new value	Set the state variable for the character agent filling the designated role to the numeric value specified by new value.
	IncrementVariable	role state variable delta	Increment the state variable for the character agent filling the designated role by the specified delta value. Delta can be positive or negative.
Connectors	ExtendInternalConnector	connector	Change the state of the connector to extended.
	RetractInternalConnector	connector	Change the state of the connector to retracted.
Objects	ObtainObject	role object	Add the object to the inner environment of the character agent filling the designated role.

Table 1. Story Engine CMAS Actions

This relatively simple set of actions is sufficient to update and maintain the character agent's state. For example, as the car-buying scene is generated, the following actions would be among those executed:

IncrementVariable (MainCharacter, bankBalance, -4000)
ObtainObject (MainCharacter, car)

The ObtainObject action will add the car object to the inner environment of the main character. In doing so, the *buy-a-car* goal will be satisfied and the state of the goal will change to *achieved*. There are additional actions associated with goals that may be executed when a goal changes state. Continuing with the car example, when the buy-a-car goal changes state to achieved, a recurring action is initiated that decrements the

bankBalance variable on a recurring basis to simulate a recurring car payment. The reactive agent controlling the character agent's goal structure manages this action.

(2) Scene Rendering Actions. Table 2 lists the set of atomic scene rendering actions with a brief description. These actions are processed by the animation engine, text-to-voice, and location generator to visually render a scene. A detailed explanation of each action is found in [Elzenga, 2001].

Action Name	Description
SetLocation	Generate a background.
PlaySound	Play an ambient sound.
StopSound	Stop playing an ambient sound.
StopAllSounds	Stop playing all ambient sounds.
PlayAnimation	Construct an animation.
PlaySentence	Generate a line of dialog.
PlayMovie	Play a movie.
SetExitCondition	Set conditions for ending animations.
AddScreenTransition	Set type of screen transition.

Table 2. Scene Rendering Actions

(3) Interaction Sequences. Interactions provide a mechanism for conditionally executing story engine and scene rendering actions based on stimuli from E_{inner}. They provide a wrapper around groups of atomic story engine CMAS and scene rendering actions so they execute in coordination. For example, when the car dealer greets the young man, a single interaction animates the car dealer, animates the young man and plays contextually appropriate dialog for the greeting. The interaction is played out against a background that is appropriate for the time of day and with landscape appropriate to the general region where the character is located. Additionally, during the interaction a connector extended on the car dealer agent senses the experience level connector on the buyer that causes the dealer to update a state variable indicating he is dealing with a naïve buyer. This state change in the car dealer impacts the remaining generation of the scene.

Interactions have three components: a control function (f_c), a set of connectors (C) and a set of actions (A) (Equation 31).

$$
\begin{array}{c}
\textit{Interaction } i = \langle f_c, C, A \rangle \\[4pt]
f_c - \textit{control function} \\
C - \textit{set of connectors } (C \subseteq C_i \cup C_e) \\
A - \textit{set of actions } (A \subseteq A_{se} \cup A_{sr}) \\
A_{se} \textit{ is the set of story engine CMAS actions} \\
A_{sr} \textit{ is the set of scene rendering actions}
\end{array}
$$

Equation 31. Interaction Definition

The control function is responsible for extending and retracting the interaction's connectors, initiating a connection to continue the interaction sequence, and controlling the execution of its atomic actions (Equation 32). The interaction's connectors, when extended in receptor mode, are used to establish the conditions under which it can execute. The interaction's connectors are a subset of the scene agent's internal and external connectors, including connectors associated by role. After the interaction executes, it extends internal connectors in stimulus mode to express what the interaction just did.

$$
\begin{array}{c}
\textit{Given,} \\[4pt]
\textit{Interaction } intx_i = \langle f_{c(i)}, C_i, A_i \rangle \\
\textit{Interaction } intx_j = \langle f_{c(j)}, C_j, A_j \rangle \\
\textit{where,} \\
C'_i \subseteq C_i \textit{ is } intx_i\textit{'s set of extended connectors (receptor mode)} \\
C'_j \subseteq C_j \textit{ is } intx_j\textit{'s set of extended connectors (stimulus mode)} \\[4pt]
\textit{then } intx_j \textit{ connects with } intx_i \ (conn(intx_j, intx_i)) \textit{ iff:} \\[4pt]
(\forall c_j \in C'_j)\, \exists c_i \in C'_i \ (conn(c_j, c_i)) \\
\scriptstyle (conn(c_j,\, c_i) \textit{ is defined by Equation 6})
\end{array}
$$

Equation 32. Interaction-to-Interaction Connection – conn($\mathbf{intx_j}$, $\mathbf{intx_i}$)

Figure 34 depicts a simple greeting, and response to a greeting, as an illustration. In this example, the main character greets the supporting character. The greet interaction is initiated from the first frame of a ticket. After the greeting executes, it extends an internal "greet" connector. The two "greet respond" interactions both require a "greet" connector, but they are conditional on the mood of the supporting character. In this case, the angry response will be executed. The response interaction will in turn, extend a "greet respond" connector, but with no interactions able to respond to the "greet respond," the interaction chain ends and the next frame of the ticket is executed.

This was a simple example to demonstrate the most basic function of interactions. In practice, the happy and angry responses would be combined into a single interaction that would execute conditionally based on the state of the supporting character. In the next chapter, a more substantial illustration is presented dealing with Army training.

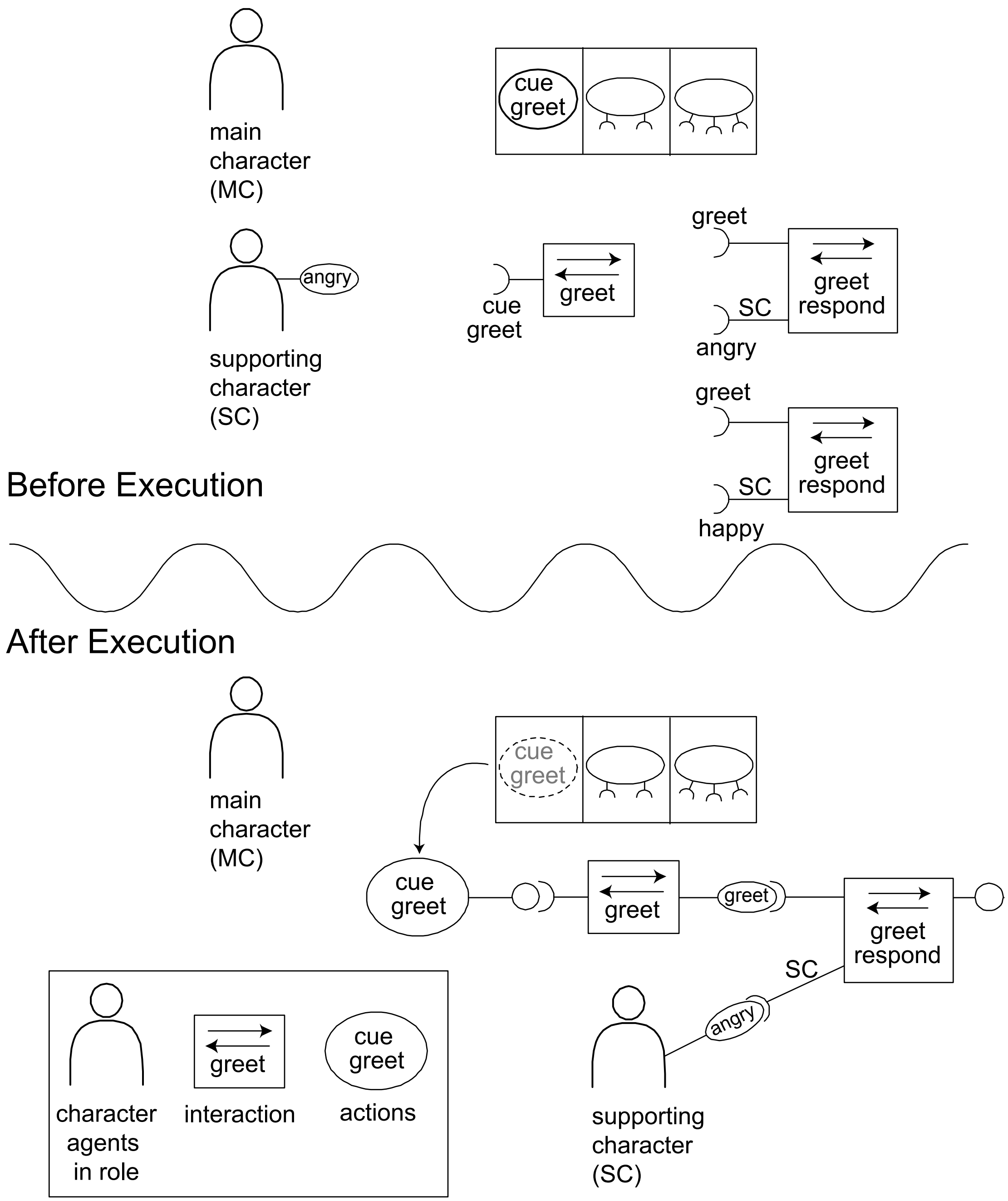


Figure 34. Greet / Response Interaction Sequence

As previously stated, when an interaction executes, it extends an internal connector. Other interactions can respond to this connector and attempt to execute. If more than one interaction tries to execute, the current approach is to randomly select the interaction. This does not occur very often since the interaction's connector set ensures only contextually appropriate interactions attempt to execute. The random selection tends to provide random variation among appropriate choices. However, the random method could be modified using a weighting function similar to the most favorable connection function (Equation 26 and Equation 30).

This process of extending connectors, and responding, creates a dynamic and contextual chain reaction of interactions not unlike the signaling cascade phenomenon found in cells (Figure 13 and Figure 25). The next section demonstrates how interactions are combined with tickets to harness the contextual chain reaction and generate scenes.

3. Character Agent to Scene Agent Connections

Story lines are generated by the story engine through a repeating process of character agents connecting with scene agents. As previously described, there is a central character agent called the main character that initiates connections with scene agents based on its goals as determined by its most favorable connection function.

Figure 35 depicts the connection process and resulting generation of a playlist, along with a post-connection view of the interaction trace. In Figure 35(a) the main character agent is evaluates its possible connections and ranks them with respect to its goals. In Figure 35(b), the agent initiates a connection with a scene agent. As a result of the connection, the "main character" role is filled. This triggers a cascade, beginning with the scene agent extending a connector to initiate a connection with an agent to fill the supporting character role. Once the roles are populated, the scene ticket begins executing (Figure 35(c)). The scene ticket initiates the process by which interactions chain together to form contextually appropriate sequences of events. As a result, the state of both the main character and supporting character are updated (story engine CMAS actions), and a playlist is generated (scene rendering actions) (Figure 35(d)). The tree structure in Figure 35(e) is an after-the-fact portrayal of the interactions (and encompassed actions) executed

by the agents as the connection progressed. This structure was generated based on the characters' goals and state, as opposed to having been laid out in advance and selected for a list of predefined plans.

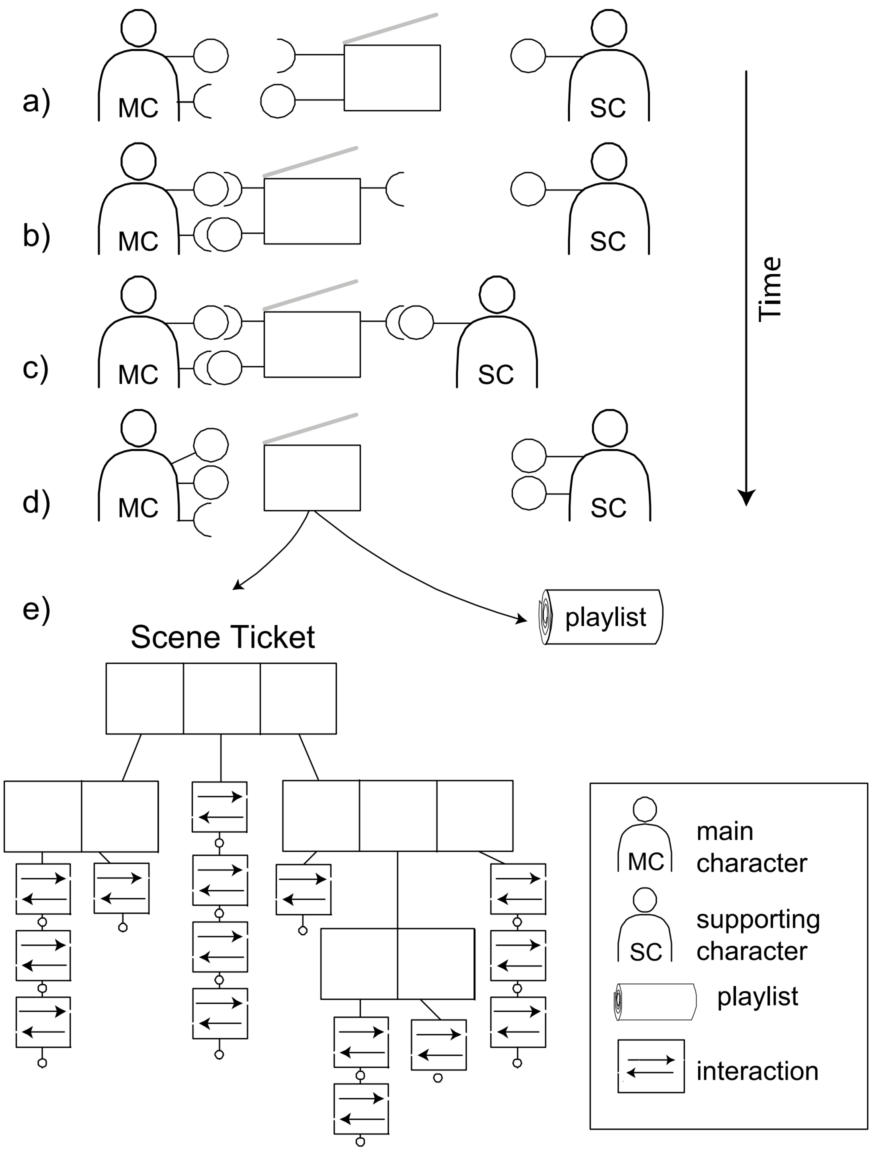


Figure 35. Character Agent to Scene Agent Connection

This next figure extends the single connection/single scene to a multi-connection/multi-scene story line generated as the main character repeatedly initiated connections in pursuit of its goals (Figure 36).

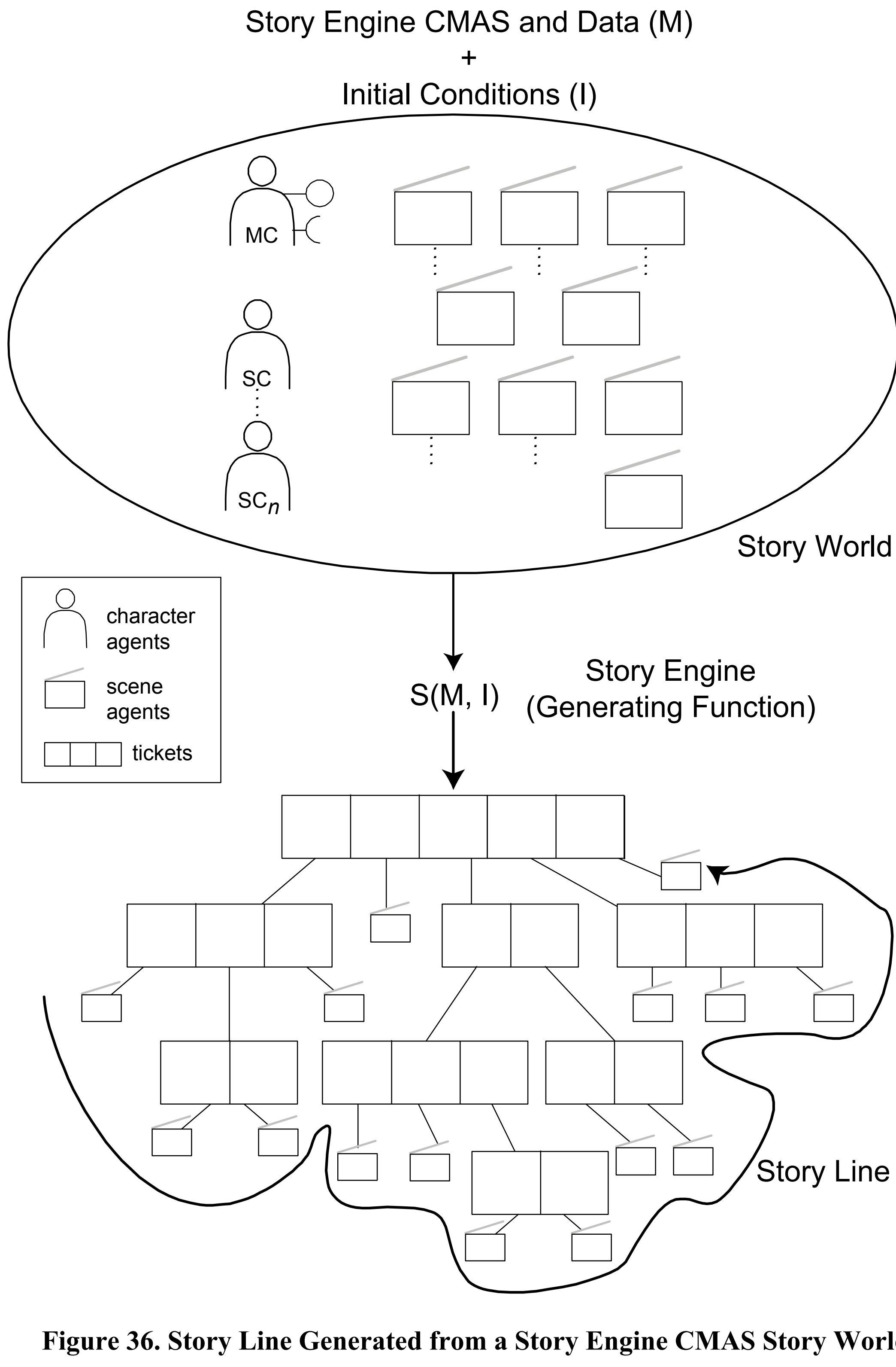


Figure 36. Story Line Generated from a Story Engine CMAS Story World

The story line in Figure 36 is tantamount to a dynamically generated plan. The scenes were generated one connection at a time as the main character agent pursued its

goals. The resultant plan evolved through the run-time binding of tickets and actions using the agent's active connectors. A depth first traversal of the ticket/scene tree explains *how* the agent achieved its goal, while the connectors provide insight as to *why* the agent generated the plan in the way that it did.

D. LOGICALLY CONNECTED, GOAL-DIRECTED AND SOUND STORY LINES

This section presents an inductive argument as proof that the story engine generates story lines that are logically connected and goal-directed. Further, it is shown that the story lines are sound with respect to the domain of interest.

1. Logically Connected and Goal-Directed Story Lines

By induction over the length of the story line, it is shown that story engine generates logically connected, goal-directed story lines.

Let $s(n)$ be a story line of length n generated by the story engine generating function $S(M, I)$ (Equation 29). $s(1)$, a story line of length one, is generated as a result of character agent a connecting with scene agent s.

The connection is established as a function of agent a's state and agent s's state, as expressed through connectors (Equation 22).

Given agent a with awareness set W (Equation 24), the set of candidate connections W' is a function of both a's state, and the state of the scene agent's in W (Equation 25).

Result 1. The set of scene agents that agent a can potentially connect with (W') is established as a function of agent a's state.

Selection of scene agent $s_j \in W'$ as the connection to initiate is the result of evaluating character agent a's most favorable connection function (*mfc*). The *mfc* is a function of agent a's goals and extended connectors (Equation 30).

Result 2. Given a set of candidate connections, agent a initiates a connection with a scene agent based on its goals, and state as expressed through extended connectors.

During the connection, sequences of interactions execute to generate a scene. Interaction sequences are self-generated based on the state of character agent a, which

initiated the connection, the state of scene agent *s*, as well as the state of the character agents filling the supporting character roles (Equation 32).

Result 3. The established connection progresses in a logically connected manner, based on the state of all agents involved (as expressed through their connectors).

From results (1), (2) and (3) above, *s(1)* is a story line that is logically connected and goal-directed.

s(1) is the product of a connection with a scene agent, where the connection is established according to a connector-based, goal-directed selection function. The connection was initiated from a candidate connection set of scene agents, where membership in the set was established as a function of agent *a*'s state and the state of the scene agents in *a*'s awareness set. Further, the connection progressed in a logically connected manner, based on the state of all agents involved.

Assuming that *s(n-1)* is a logically connected and goal-directed story line, *s(n)* is likewise logically connected and goal-directed.

Since *s(n)* is generated in sequence, the story line can be decomposed into the first n-1 scenes concatenated with a story line of length one:

$$s(n) = s(n\text{-}1) + s(1).$$

If *s(n-1)* is logically connected and goal-directed and the same is true for *s(1)*, then *s(n)* is a goal-directed and logically connected story line of length *n*.

2. Sound Story Lines

The story engine is a simulation engine. As such, it is used to evaluate software implementations of computational models. The computational model, when combined with a data set, is an abstraction of a real world system. The story engine CMAS is one such computational model that formulates the real world system in terms of character agents, scene agents, tickets, connectors, interactions, and story engine actions.

Equation 29 established the story engine as a generating function (*S*) that takes as input a story engine CMAS (*M*), along with a set of initial conditions (*I*), and generates output that includes a story line *L*. This can be expressed as $L \in S(M, I)$.

The story engine is said to be *sound* with respect to a domain of interest if $S(M, I)$ generating a story line L, implies that L is a story line from the domain of interest. Let D be a domain of interest, then $S(M, I)$ is *sound* with respect to D if

$$L \in S(M, I) \rightarrow L \in D.$$

The story engine CMAS is a purely data-driven architecture, where the character agents, scene agents, tickets, etc., are instantiated solely from data, not by the CMAS architecture. If D is a real-world domain of interest, let X be a story engine CMAS data set used to model D, and M_x be the story engine CMAS instantiated from X. X describes the domain D in terms of character agents, scene agents, tickets, connectors, interactions, and story engine actions. The CMAS data representation of D is a model, and as such, it is an abstraction of the real world domain. Since X is an abstraction of D, let D_x be the actual domain modeled by X.

In the previous section, it was established that the story engine generating function generates story lines that are connected logically based on agent connections and connector state. Agent connections are established, and connector state is maintained, by the story engine CMAS constructs, strictly as defined by the data. Therefore,

$$L \in S(M_x, I) \rightarrow L \in D_x,$$

and $S(M_x, I)$ is *sound* with respect to D_x.

If X accurately represents the temporal, structural, and procedural relationships and constraints of D, then $D_x \subseteq D$.

$$\text{If } (D_x \subseteq D) \wedge (L \in D_x), \text{ then } (L \in D).$$
$$\therefore (D_x \subseteq D) \rightarrow [L \in S(M_x, I) \rightarrow L \in D].$$

So, if the CMAS data set X accurately describes the domain of interest D, then the story engine $S(M_x, I)$ is *sound* with respect to D.

E. RUN-TIME ANALYSIS AND SCALABILITY OF STORY LINE GENERATION

Story lines are generated on a connection-by-connection basis as initiated by the main character agent. In this section, it is shown that for a given story engine CMAS M, there are a factorial number of possible story lines. In addition, a run-time analysis is presented that shows a story line of length k is generated in $O(kca)$ where c is the number of connector types and a is the number of agents in the story engine CMAS.

1. Run-time Analysis

Let M be a story engine CMAS with a agents (a_1 scene agents and a_2 character agents) and a connector set with c connector types. At any given step of the story line, the main character agent (a_{mc}) could have at most c connectors extended and a_1 scene agents in its awareness set. If a_{mc} has c connectors extended, then to determine if it can connect to an agent $a_i \in a_1$, it must compare each of its c connector types with the matching type (if extended) on a_i. This comparison is a constant time $O(1)$ operation. So, in the worst case, a_{mc} is able to evaluate its connection with a_i in time $O(c)$. If there are a_1 agents, each with c connectors, then the main character requires at most ca_1 comparisons to determine which scene agent to connect with.

Once a connection is established between the main character agent and the scene agent, the scene agent must then establish connections with the supporting character agents. There are at most a_2-1 character agents to fill the supporting character roles (one of the characters is the main character). The scene agent must make at most $c(a_2\text{-}1)$ comparisons to fill its supporting character roles. Therefore, at each step of the story line generation, the number of comparisons to initiate all required connections and begin the scene generation process is $c(a_1) + c(a_2\text{-}1)$, which is equal to $c(a\text{-}1)$, or $O(ca)$.

Scenes generate by executing a subset of the scene agent's set of internal interactions. Since each interaction can execute at most once, this process occurs in constant time. Therefore, a single element of the story line is generated in $O(ca)$, and a story line of length k, is generated in $O(kca)$, where c is the number of connector types and a is the number of agents in the story engine CMAS.

2. Story World Dimension

Let $N(M)$ represent the number of story lines possible from the story engine CMAS M and let $N(M, k)$ be the number of possible story lines of length k. It is shown that $N(M)$ is $O(a_1!)$, where a_1 is the number of scene agents in M.

Given the set of a_1 scene agents, if it is assumed that each scene agent can be connected with at most once, and that all orderings and combinations are possible, then the number of possible story lines of length k ($k \leq a_1$) is equal to the number of k-*permutations* from a set of a_1 distinct scene agents, denoted $P(a_1, k)$. This is, however, an

upper bound because the *actual* number of possible story lines is a function of the temporal constraints on their ordering and combination as implied by tickets and connectors. Therefore,

$$N(M,k) \leq P(a_1,k) .$$

Summing across story lines from length 1 to a_1 yields;

$$\begin{aligned}
\sum_{k=1}^{k=a_1} N(M,k) &\leq \sum_{k=1}^{k=a_1} P(a_1,k) \\
&= \sum_{k=1}^{k=a_1} \frac{a_1!}{(a_1-k)!} \\
&= a_1! \sum_{k=1}^{k=a_1} \frac{1}{(a_1-k)!} \\
N(M) &= O(a_1!)
\end{aligned}$$

Therefore, given a story engine CMAS *M*, there are a factorial number of possible story lines, and for any given story line of length *k*, the story engine generates the story line in time $O(kca)$, where *c* is the number of connector types and *a* is the number of agents in the story engine CMAS.

F. SCENE RENDERING SUBSYSTEM

The Interactive Story Generation System (ISGS) combines the story engine with a scene rendering subsystem (SRS) and a user interface to present and control interactive stories (Figure 29). The SRS and graphical interface were developed independent of the story engine by programming teams supporting the *America's Army: Soldiers* project. In its current state, the ISGS allows for user intervention between scenes but not during the scenes, resulting in interactive stories, but not interactive scenes. While the story engine is capable of supporting intra-scene user intervention, the SRS, at present, is only able to render scenes in their entirety.

The SRS provides an after-the-fact view of the actions that occurred during the agent-to-agent connection. The story engine and SRS interface via a playlist generated by the scene agent. The playlist is a compilation of SRS specific commands used to render the scene. The SRS is comprised of three integrated components; an animation engine that controls the characters' movements on screen, a location generator that creates contextually appropriate photo-realistic backgrounds and a text-to-voice system that combines prerecorded dialog pieces in accordance with a narrative grammar.

1. Animation Engine

The animation engine operates on an event-based, time relative structure for media playback. In this scheme, animations are not scheduled for play at strict times or for a predefined number of frames. The length of the animation is dependent on outside cueing, such as sound files beginning to play, sounds files completing, or keystrokes and mouse clicks. In the opening of the car lot scene, the car dealer greets the potential buyer. From the SRS perspective, this entails animating the car dealer speaking, animating the buyer listening and playing a greeting dialog. The greeting should be contextually appropriate, so the exact series of sound files to be played is not set until the greeting interaction is executed. As a result, the length of time the greeting dialog plays is variable. In order to accommodate varying length dialog, the character animations must play in synch with the spoken dialog. Sustained animation, synchronized to outside cueing, has been used successfully in a number of applications including the Microsoft Office 97 office assistant [Kessler and Kilgore, 1997].

The SRS animation engine constructs animations from individual frames of digital imagery. Actors are filmed against a blue screen going through a range of motions and gestures. The raw media is processed and a portion of it is used to construct a database of animation frames. Much like a cartoon animator brings a character to life through a sequence of still images, the animation engine uses the same technique to animate a character agent using individual frames of digital imagery. For a given animation action, the engine selects a set of prescribed frames from the database and plays them in accordance with action-specific ordering and branching rules to generate the animation. Each character has a fixed database of frames. But, once the frame set is established and cataloged, the frames can be reused in many different animations. A relatively modest set of frames can be used to produce hundreds of on-screen actions. A detailed description of the animation engine can be found in [Elzenga, 2001].

2. Location Generator

Locations are the stage on which the story action takes place. From a single data set, the story engine is capable of generating hundreds of scenes being played out in many different settings (locations). Rendering the scenes requires the ability to situate

stories and characters in contextually appropriate and believable locations. The location generator is responsible for constructing backgrounds that are appropriate for the given scene and also fit the character. It manages a database of reusable media pieces to construct the locations at run-time (Figure 37).

A typical location consists of six to ten media pieces that include a sky, mid-ground and foreground, context-specific props, and character animations. If the character is at a beach in southern California, the background should reflect a smooth sandy beach. On the other hand, if the same scene occurs while the character is in New England, the background should include a rocky shoreline. The location generator populates location templates at run-time with media that is appropriate for the character's current state. The location generator makes use of the character agent's connectors when populating the template. Details of the instantiation process can be found in [Elzenga, 2001].

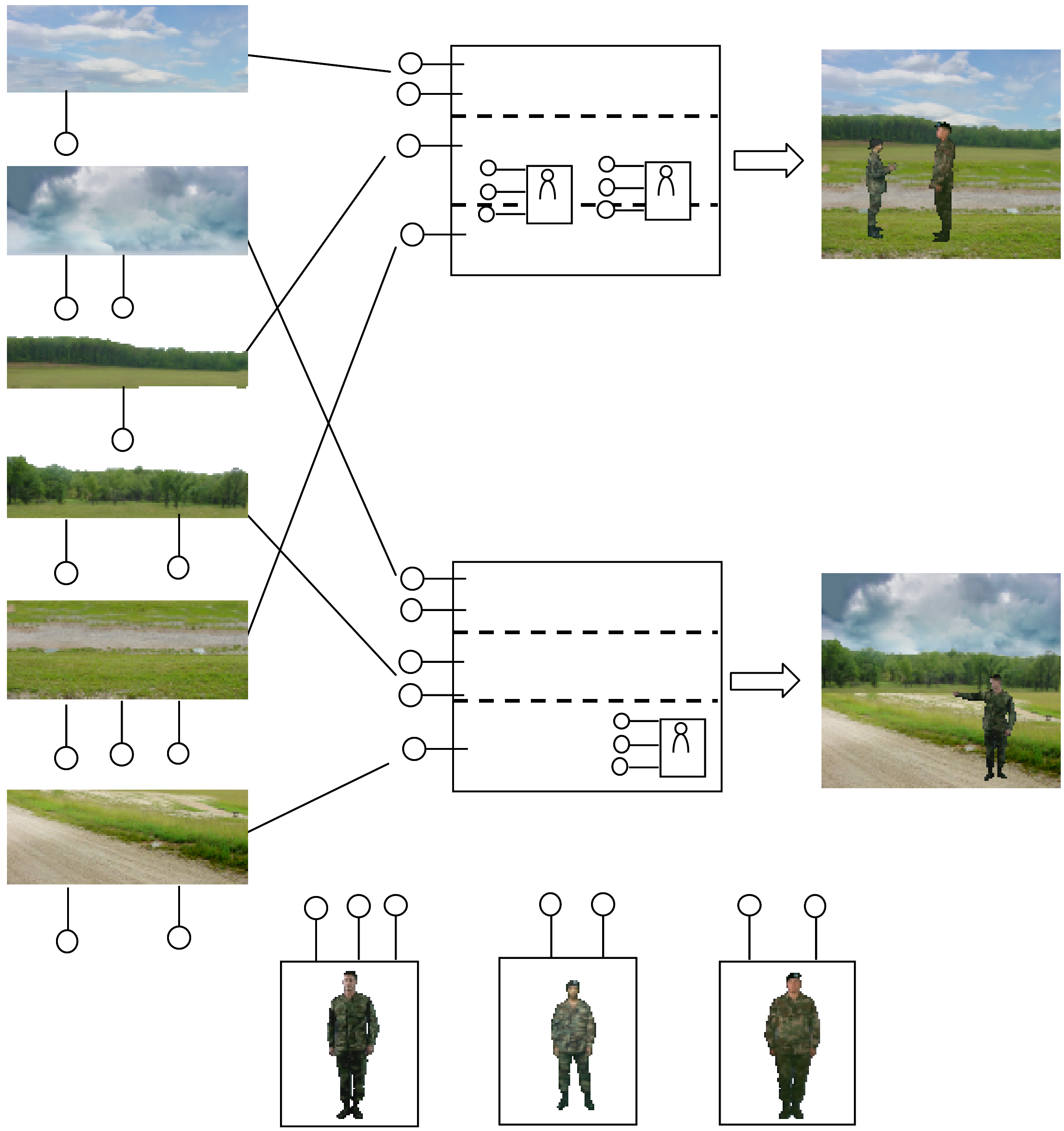


Figure 37. Location Templates

3. Text-to-Voice

The process for constructing character dialog is similar to constructing locations. With locations, a template is used to guide the selection of images used to create the background. In the case of the text-to-voice system, a generative grammar is used to define sentence structures. These structures included variables appropriate to the scripts and the domain. Voice actors record full sentences that are then split into atomic pieces

and placed in a database. The text-to-voice system's sentence generator parses sentence definitions and constructs the dialog from the database of atomic speech files.

Figure 38 shows an example of a greeting sentence for the car dealer scene. Based on the time of day, gender and age of the customer, and the dealer's analysis of the customer's experience level, an appropriate greeting is constructed. The greeting can range from "Good morning ma'am, can I help you?" to "Hi there young man. Welcome to Billy Bob's. This is your lucky day. I just received a car that you are going to love." [Elzenga, 2001] provides a full description of the text-to-voice system and its integration with the animation engine.

<greeting> :: <salutation:parameters><title:parameters><welcome:parameters>

<salutation>[parameters: time of day]
Good morning [between 0800 and 1200]
Good afternoon [between 1200 and 1800]
Good evening [between 1800 and 2200]
Hello [random]
Hi there [random]

<title>[parameters: gender, age (as determined by car dealer)]
sir [male, 30 or over]
ma'am [female, 30 or over]
young man [male, under 30]
young lady [female, under 30]

<welcome>[parameters: buyer's experience level as determined by car dealer]
Can I help you? [experienced]
What can I do for you? [random]
Welcome to Billy Bob's. This is your lucky day. I just received a car that you are going to love. [naïve]

Figure 38. Text-to-Voice Generative Grammar

G. GRAPHICAL INTERFACE

The graphical interface provides the interactive link between the story engine and the user. As a simulation engine, the story engine is capable of generating story lines without intervention. However, by allowing the user to interact with the agents, the story engine is able to present a story line that is highly personalized. In the ISGS MVC architecture, the interface augments the automated control provided by the story engine CMAS with a "manual" control mechanism. The level of interaction is application

dependent. The user can take on a highly involved first person role as a character in the story, or step back and influence the story via high-level adjustments to the story world's relations and laws. In *America's Army: Soldiers*, presented in the next chapter, the user guides their character by adjusting the character's goals and values, but they do not have direct control over the character's actions. The character remains an autonomous agent.

H. SUMMARY

The ISGS structures and presents interactive stories through an instance of a CMAS called a story engine CMAS. The story engine CMAS is a data-driven architecture that when combined with a domain specific data set, generates dynamic, goal-directed story lines. The SRS processes playlists generated by the story engine's scene agents, to present a view of the story as it unfolds. Through a graphical interface, a user is able to interact with the agents and objects of the CMAS story world.

The story engine's story lines are assembled on a connection-by-connection basis as a by-product of character agents pursuing their goals. The generation of a story line is equivalent to the construction of a historical tree that records the derivation of the agent's plan to achieve its goal. A depth-first traversal of the tree provides an explanation of how the agent reached its objective.

VII. CMAS DATA SET IMPLEMENTATION – AMERICA'S ARMY: SOLDIERS

A. INTRODUCTION

The story engine was fielded as the underlying simulation engine in an interactive, story-based role-playing game for the U.S. Army. This chapter introduces the *America's Army* project, and describes the story engine CMAS data set used to interactively generate story lines focused on army career progression. The tickets, connectors, scenes, and characters, including their goals and personalities, are presented. The *America's Army: Soldiers* project is offered as a proof-of-concept of this dissertation research.

B. AMERICA'S ARMY: SOLDIERS

America's Army is a suite of two applications developed for the U.S. Army intended to provide young people with accurate, easy-to-assimilate information about the Army so they can develop a better understanding of army life and available opportunities. The two applications, *America's Army: Operations* (*AA: Ops*) and *America's Army: Soldiers* (*AA: Soldiers*) offer a realistic portrayal of Army training, missions and values.

AA: Ops is a first-person perspective gaming environment that demonstrates life in the infantry. The experience begins with required individual training, such as basic rifle marksmanship and Basic Combat Training (BCT). Once qualified, the player is able to take part in multi-player missions. *AA: Ops* focuses primarily on combat training and tactical missions. While these are critical, they represent only a small part of army life.

*AA: Soldier*s is a sophisticated role-playing game that provides a look at Army personnel and career opportunities by allowing the player to guide a character through a career in the Army. It presents the Army through stories that are sensible yet surprising, interactive, and non-repeating. In order to visually portray Army bases, offices, barracks and facilities as accurately and faithfully as possible, digital photographic and video imagery was chosen as the display medium for the Scene Rendering Subsystem.

A story engine CMAS data set was constructed to accurately capture the Army domain in terms of career progression, general training and specialty schools, as well as on and off-duty life.

Interaction in the game commences at character definition time, and continues throughout the game by adjusting the character's "core values" and goals. The player cannot exert direct control over the character, i.e., cannot make a specific decision for the character. The character is an autonomous agent; as such, the player must guide their character through the story world by intermittently adjusting its goals and values.

C. INTERACTIVE STORY GENERATION SYSTEM DATA

The Interactive Story Generation System data is partitioned into two sets, data used to define the *AA: Soldiers* story world, and data necessary to visually render the scenes (Figure 39). The story world is defined by a story engine CMAS data set comprised of three primary parts: character agent definitions, scene agent definitions, and object definitions. Connectors from the character agent's inner environment and goal set, as well as a small collection used by the scene agents to describe the setting of the action, are combined to make up the *AA: Soldiers* connector set. This set is described later in the chapter. The Scene Rendering Subsystem is instantiated from data that is specific to each of the modules: speech, animations and locations.

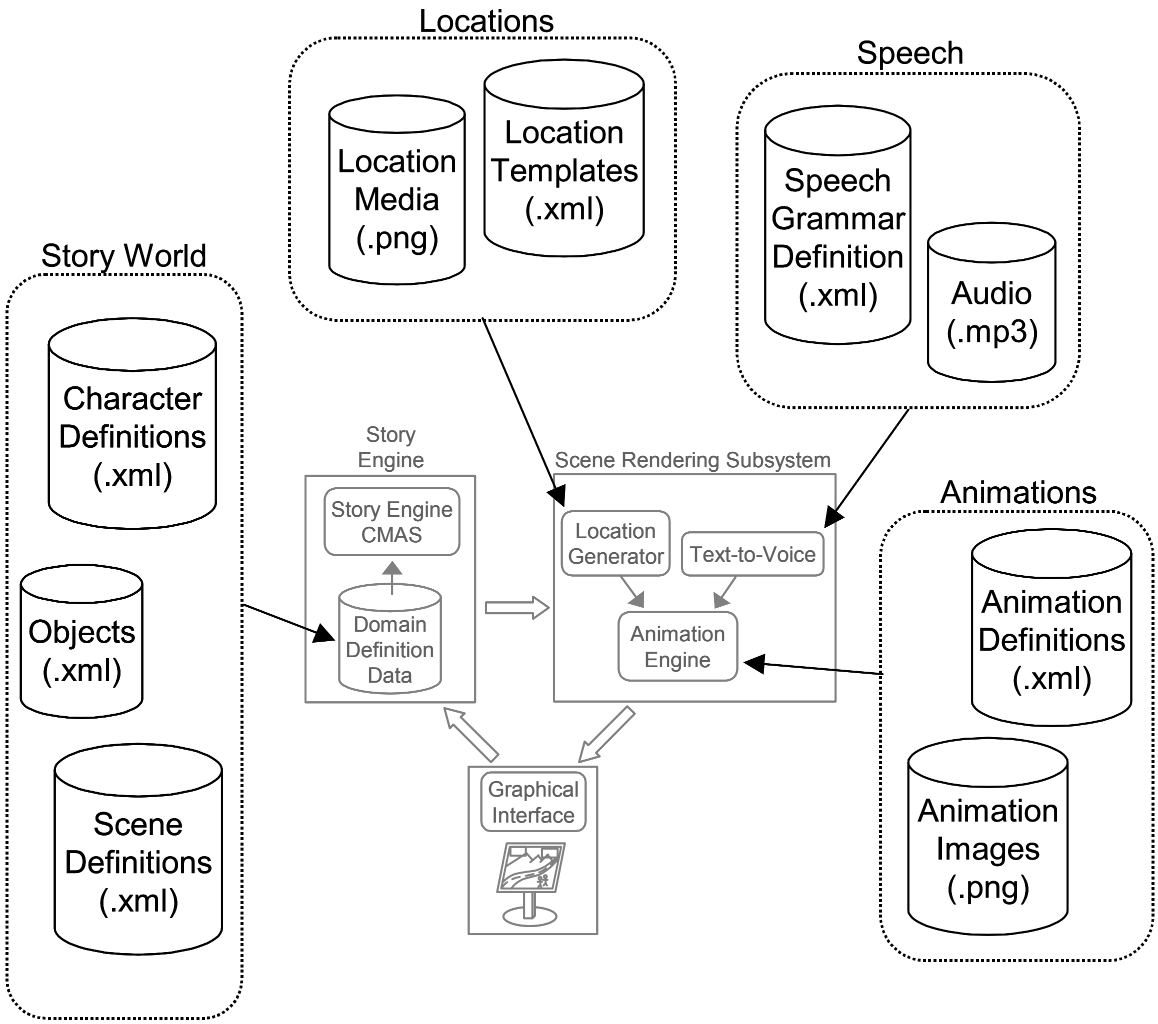


Figure 39. AA: Soldiers Interactive Story Generation System Data Set

D. CHARACTER DEFINITIONS

Characters in *AA: Soldiers* are defined according to their personality, aptitude and goals. Personality is delineated in terms of the Army's seven "core values" [U.S. Army, 1999], while aptitude is measured according to six "resource" levels. A character is also bestowed with a set of goals, which are managed by the player. In addition, character agents are instantiated with a set of tickets and actions reflective of Army career progression. In terms of story worlds and story lines, the main character's personality, aptitude and goals establish the initial conditions for the story world. These attributes also combine to drive the character's behavior and, ultimately, the story line. While tickets provide general story direction, i.e., Army career progression, the character's

goals influence its path through the story world. The character's aptitude, as determined by its resources, determines how quickly the character progresses toward achieving its goals. They provide fuel for the character. Finally, personality establishes how steadfast the character is in working towards its goals. Low core values can result in the character losing focus and being led astray.

This application is designed to introduce young people to all aspects of the Army, including providing them with a view of what the Army expects of its recruits and qualified soldiers. Misaligned values, resources and goals can attract the character to scenes that exploit out-of-balance conditions.

1. Actors, Characters and Roles

The SRS uses digital imagery and recorded audio to animate the characters and construct their dialog. As such, a set of "actors" were filmed and photographed against a blue-screen, and recorded in a sound studio to generate the required images and audio. In conjunction with the actors, the story engine maintains a list of recognized "roles." These roles include main character, supporting character, drill instructor, car salesman, and so on. Associated with a role is the requirement to take certain physical actions, i.e., look left, point right, kneel down, etc. Roles such as main character and supporting character are very general and require a wide range of actions. Other roles are more specific, such as the salesman, and require a smaller range of actions. Since not all actors have the digital frames to support every possible action, an actor definition file is used to delineate the roles the actor is capable of filling. In the case of the main character "role," the player selects an "actor" and sets values, resources and goals to define their "character." Figure 40 depicts the screen used to define the player's "character," which fills the main character "role." The remaining actors are mapped to characters by the story engine.

Figure 40. AA: Soldiers Character Definition Screen

2. Core Values and Resources

A character's personality is defined in terms of the Army's seven core values: loyalty, duty, respect, selfless service, honor, integrity, and personal courage (Figure 41). There is a state variable corresponding to each of these that ranges from zero to 100. Associated with each variable, is a connector whose control function graduates the variable into five levels ranging from "low" to "high." As the variable changes value, the connector changes state accordingly. When a new game is initialized, the player defines a character, which includes distributing a limited budget of "value points" among the seven core values. As the game progresses, additional points are accumulated, or lost based on the character's actions. The player intervenes as desired to distribute newly accumulated points, or readjust the current cache of points.

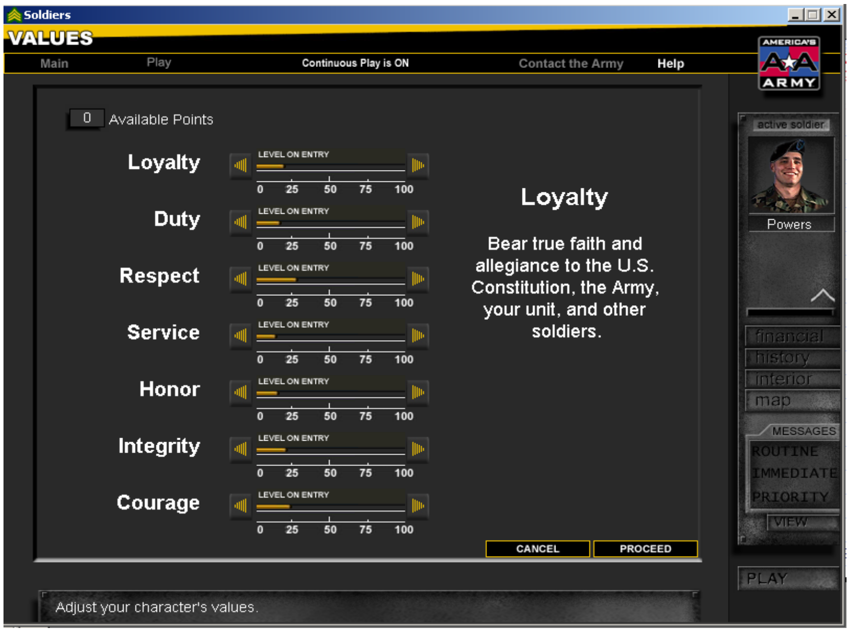


Figure 41. Character Personality Defined by Army Core Values

Six resources are used to drive the character: energy, skill, strength, knowledge, financial, and popularity (Figure 42). Like the core values, the connectors associated with each of these are graduated from "low" to "high." However, unlike core values, the player cannot directly adjust the resources. Resources increase and decay based on the character's actions and achievements.

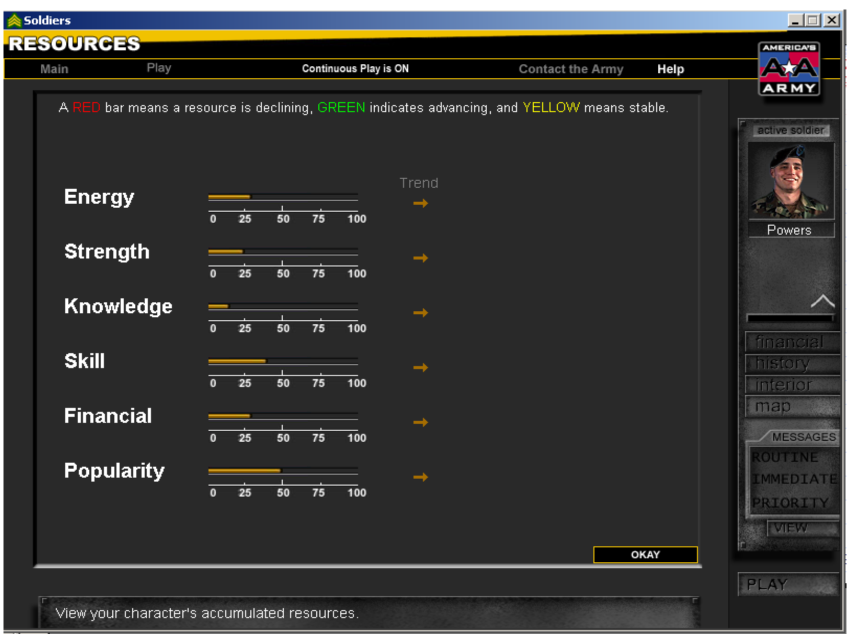


Figure 42. Character Aptitude Expressed in Terms of Resources

3. Goals

AA: Soldiers' goal set is widely varying, including goals targeted at succeeding in the Army, and others representative of the target age group of 18 to 24 year olds. There are 60 goals in the application that deal with personal and professional growth (Table 3). The goal set was arrived at through joint consultation between the *AA: Soldiers* project manager and Army's project sponsor. Personal goals range from party, date, and buy a car, to establish credit, college degree and time management. The professional goals cover the major areas of career, soldiering and maturing. Career goals deal with solving work problems, creating solutions, and multi-tasking. Maturing goals cover aspects such as dealing with authority, following directions and paying attention to detail. Soldiering goals focus the character on developing professional skills required of a soldier, such as marksmanship, map reading and meeting physical training requirements.

Have Fun	Party	Be Cool
Minimize Work	Pay Attention to Detail	Deal with Authority
Teamwork	Work Hard	Study Skill
Advanced Study Skills	Oral Communications	Good Vocabulary
Writing Skills	Advanced Writing Skills	Physical Exercise
Physical Training (PT) Program	Amateur Sports	Marksmanship
Advanced Marksmanship	Personal Luxuries	Stereo
TV	Computer Games	Home Entertainment Center
Car	Sports Utility Vehicle	Luxury Car
Date	Relationship	Marriage
Baby	College AA Degree	College Bachelors Degree
College Graduate Degree	Professional Certificate	Qualify PT Run Time
Qualify PT Push-ups	Qualify PT Sit-ups	PT Badge
Advanced PT	Extreme PT	Map Reading
Visualize Map	Reading Skills	Reading for Speed
Advanced Reading Skills	Time Management	Multi-tasking
Meet Deadlines	Follow Directions	Solve Work Problems
Create Solutions	Foreign Language	Establish Credit
Pay Bills	Savings Account	Manage Money
Live Off Post	Read Technical Manuals	Trouble Shoot Problem

Table 3. Character's List of Potential Goals

Unlike most agent-based simulations, where the agent maintains its own goal priorities, in this application the player is responsible for selecting and prioritizing the their character's goals. The player selects and prioritizes up to five goals (Figure 43).

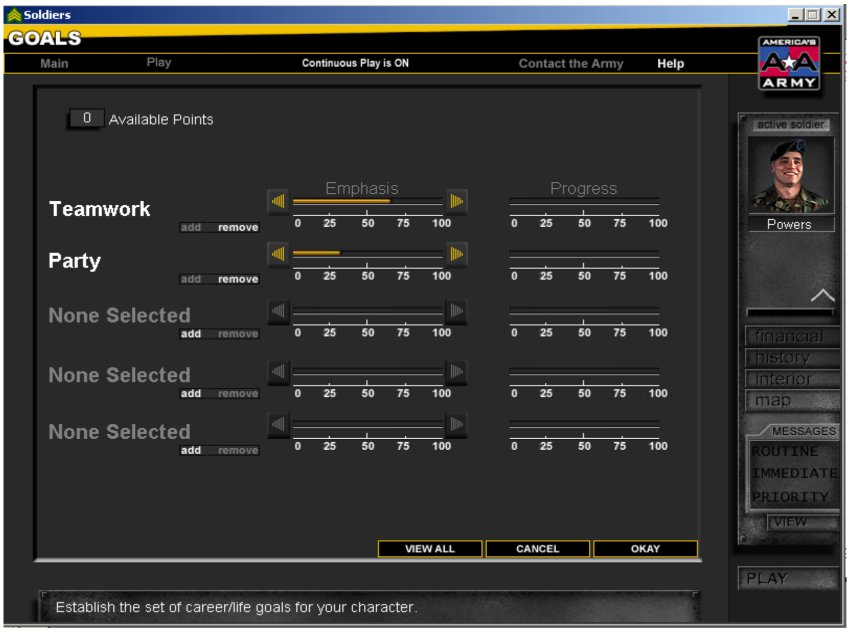


Figure 43. Goal Emphasis and Progress

Goals have preconditions for selection, requirements that must be met to achieve the goal, and a set of results or effects of achieving the goal (Figure 44). Preconditions are defined by a set of prerequisite goals that must be achieved before the goal is available to be selected. This relationship creates a "goal hierarchy," a portion of which is shown in Figure 45. The result of achieving a goal includes unlocking new goal opportunities, adding to, or subtracting from resource levels, and strengthening or weakening values. The goal's progress (measure) and emphasis (weight) are used to drive the player's experience as they progress through the game.

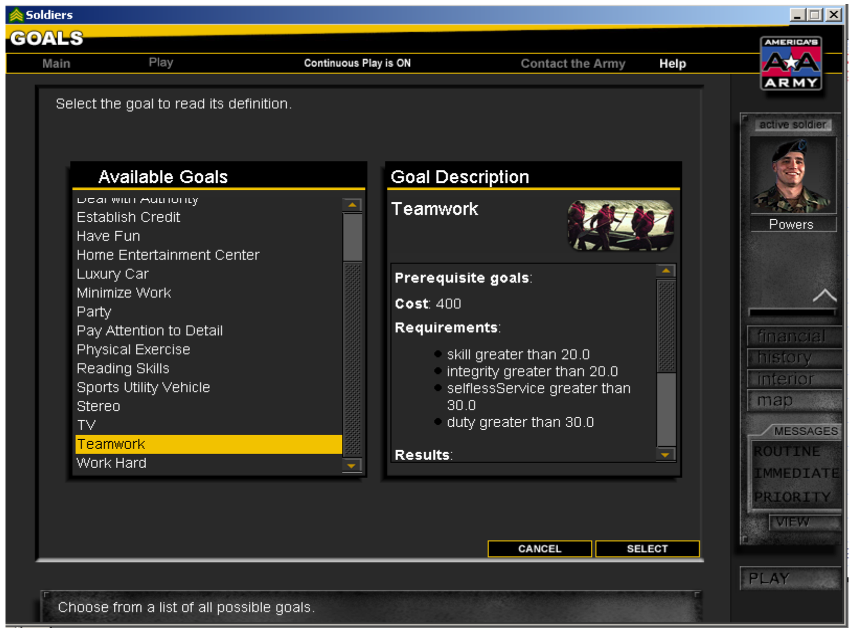


Figure 44. Goal Description Including Prerequisites, Requirements and Results

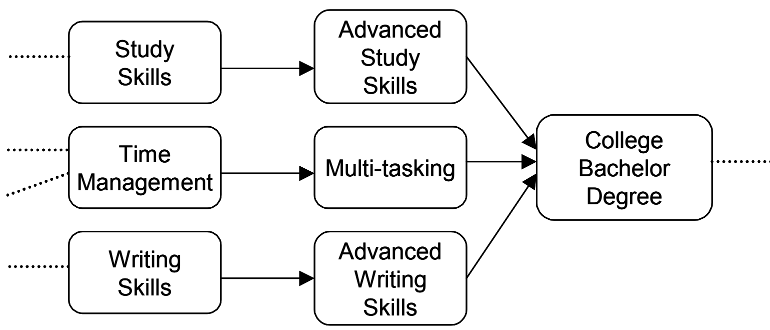


Figure 45. Goal Hierarchy Formed from Goal Prerequisites

4. Tickets: Army Training Progression

Army training progression, particularly in the early stages of a soldier's career, is highly structured (Figure 46). This structure is reflected in the tickets defined for the character agents. Table 4 provides a list of the primary tickets used to structure the character's progression through the initial stages of Army training. These relatively simple tickets are sufficient to structure approximately the first nine months to a year of a recruit's life in the Army. At the current time, this period of training is the primary focus of *AA: Soldiers*. The list described in Table 4 is only a partial list, in that for each of the Army occupational specialties, there are tickets that capture the unique elements of the training regimen. Basic Combat Training (BCT) and the initial stages of One Station

Unit Training (OSUT) are similar for all career specialties. However, Advanced Individual Training (AIT) and the later stages of OSUT are specific to the operational specialty selected.

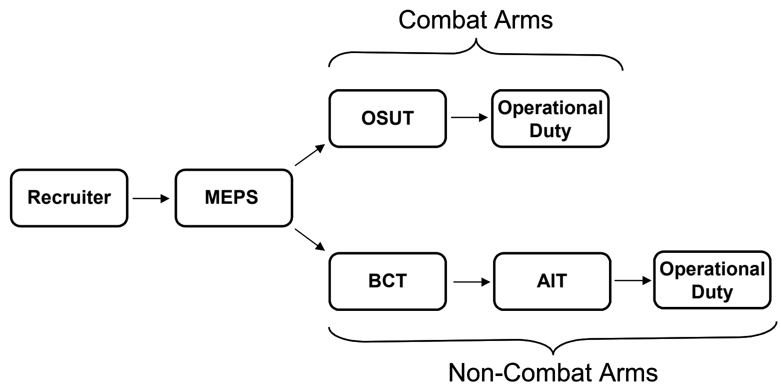


Figure 46. U.S. Army Career Training Progression

Ticket	Description
MEPS	Military Entrance Processing Station: An initial processing stage required for all military recruits.
BCT	Basic Combat Training: Initial training for recruits entering non-combat arms specialties.
AIT	Advanced Individual Training: Follow-on training to BCT that is related directly to the recruit's chosen specialty.
OSUT	One Station Unit Training: Basic and specialty training combined into a single program for combat arms specialties.

Table 4. Army Training Progression

Once a character enters their first duty assignment, the career progression possibilities begin to expand. Just as with any profession, future opportunities are normally a function of past performance. In the Army, there are a number of educational and professional opportunities that are based on performance. Some of these include Airborne training, Ranger training, and Special Operations. These schools, and paths to advanced designations, are structured as tickets, with connectors used to enforce the prerequisites in terms of core values (integrity, personal courage,…), resources (knowledge, strength,…) and experience, such as AIT complete and Airborne qualified.

E. CONNECTOR SET

This section describes the *AA: Soldiers* connector set. It is a compilation of connectors used to capture character state, Army training progression, and backdrops for

the scenes (Table 5 and Table 6). As indicated earlier, an *AA: Soldiers* character is described according to their core values, resources and goals (Table 5). Army career progression is represented by occupational specialty, post (Army base), career phase, and duty phase (Table 6). Career and duty phase combine to model the stages of an Army career. Within the career phases, particularly the training segments, there is a well-defined structure that is represented by the duty phase connector. Figure 46 depicts the career phases from initial enlistment up through the soldier's first operational duty assignment. Sixteen military occupational specialties (MOS) and fourteen posts are represented in the game. Finally, there are connectors that assist in establishing the stage setting (location) where the action takes place and scene plays out (Table 6).

	Connector Type	Connector Values
Core Values	Loyalty	Low, MediumLow, Medium, MediumHigh, High
	Duty	"
	Respect	"
	Selfless Service	"
	Honor	"
	Integrity	"
	Personal Courage	Low, MediumLow, Medium, MediumHigh, High
Resources	Energy	Low, MediumLow, Medium, MediumHigh, High
	Strength	"
	Knowledge	"
	Skill	"
	Financial	"
	Popularity	Low, MediumLow, Medium, MediumHigh, High
Goals	Have Fun	Selectable, Selected, Achieved
	Party	"
	⋮	"
	Marksmanship	"
	Teamwork	"
	Manage Money	Selectable, Selected, Achieved

Table 5. Connectors Describing Character Personality, Aptitude and Goals

Career Progression Connectors				Loction Connectors	
MOS	Post	Duty Phase	Career Phase	Location Group	Place
Infantry	Ft. Benning	Arrival	Recruiter	Formation	Gym
Combat Engineering	Ft. Bliss	Indoctrination	MEPS	Work Detail	Office
Field Artillery	Ft. Bragg	Phase 1	BCT	Training Station	Barracks
Air Defense Artillery	Ft. Eustice	Phase 2	AIT	Field Training	Apartment
Special Forces	Ft. Gordon	Phase 3	OSUT	Off Duty	Game Room
Armor	Ft. Hood	Qualified	Operational Duty	Classroom	Weight Room
Signal Operations	Ft. Huachuca	Graduation	Discharged Honorable	PT Field	Garage
Electronic Maintenance	Ft. Jackson		Discharged OTH	Rifle Range	Aircraft hangar
Chemical	Ft. Knox		Retired		Parking Lot
Ammunition	Ft. Lee				Obstacle Course
Administration	Ft. Leonard Wood				Commissary
Petroleum and Water	Ft. Sam Houston				PX
Medical	Ft. Sill				Chapel
Supply and Services	Ft. Wainwright				Bank
Military Police					Car Lot
Military Intelligence					Lake
					Jail
					Pawn Shop
					Barracks
					Restaurant
					Airport

Table 6. Connectors Describing Army Career Progression and Locations

F. SCENE DEFINITIONS

Scenes are defined in terms of interactions, tickets, internal connectors and roles (Figure 33). This section demonstrates, by example, the scene generation process, and how the outcome of the scene is a function of the main character, as well as the characters filling the supporting roles. The following portrays the main character attempting to qualify in the run portion of the physical training (PT) test during phase two of basic training.

1. Exemplar Scene: Physical Training

The outcome of a scene includes not only the rendering and result (quit, fail or complete the run), but also the changes that occur to the characters participating in the scene. Figure 47 depicts the in-process connection between the main character and PT scene agent, with the supporting character connection already established. Table 7 catalogs the interactions from the PT Scene agent, including the connectors (internal and external) and actions. The scene progresses according to the three-frame scene ticket in the scene agent. The first and last frames deal with setting and clearing the stage ("Set Stage" and "Clear Stage" respectively). These two actions are not described since they deal almost exclusively with the Scene Rendering Subsystem.

The main action of the scene is initiated in the second frame of the scene ticket with the "Start Run" interaction (Table 7). This scene is meant to challenge a character with marginal personal courage and strength; characters with medium to high personal courage and medium to high strength complete the run without problem (Table 7: Finish(1)). A main character with low personal courage or low strength could possibly complete the run, provided they get some encouragement from the supporting character (Table 7: Encourage, Finish(2)). On the other hand, a supporting character with low selfless service can discourage the main character causing them to quit (Table 7: Quit).

As the scene generation progresses, the values and resources of the characters are updated, leaving them in a new state once the agents disconnect. In this example, the supporting character discourages the main character, causing the main character to quit. The result is lower strength and personal courage on the part of the main character and lower selfless service and loyalty on the part of the supporting character.

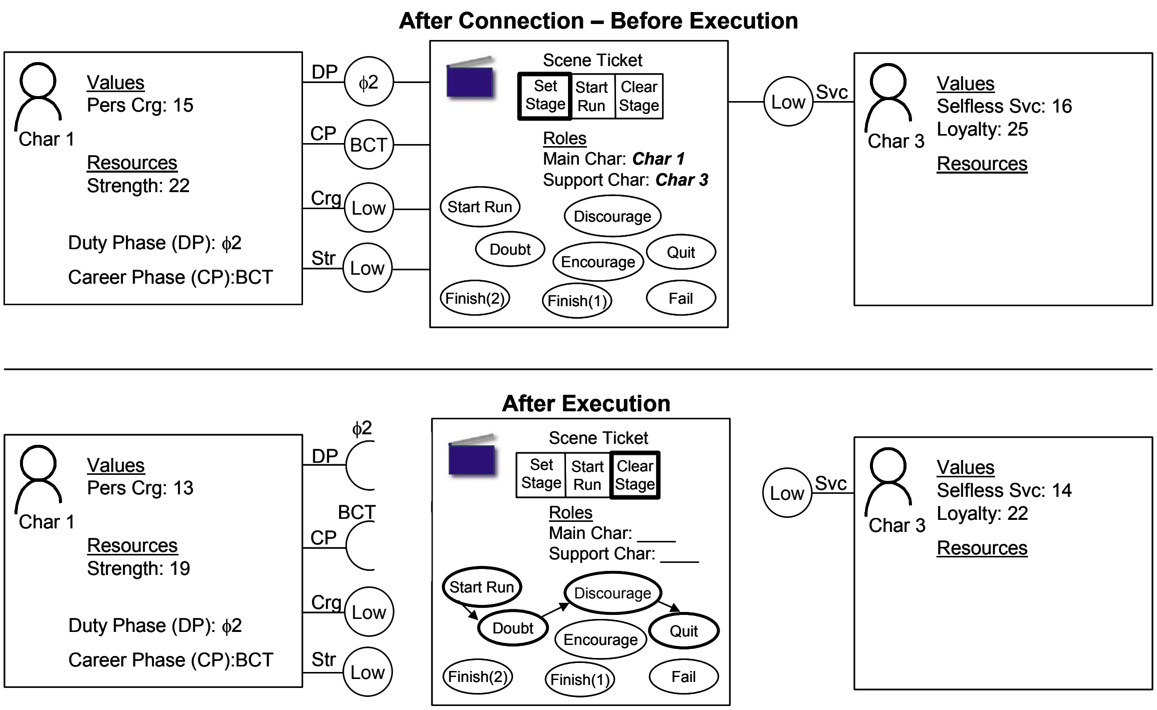


Figure 47. Basic Physical Training: Run Qualification Scene

Interaction	Connectors	Actions
Start Run	IC: CueStartRun	IC: Run Started
Finish(1)	IC: Run Started MC: Strength (Med - Hi) and Personal Courage (Med - Hi)	IC: Finish MC: Personal Courage +2 Strength +2
Doubt Ability	IC: Run Started MC: Strength (Low) or Personal Courage (Low)	IC: Doubt
Encourage	IC: Doubt SC: Self Svc (Med - Hi)	IC: Encourage SC: Loyalty +1 Selfless Service +1 MC: Personal Courage +2
Discourage	IC: Doubt SC: Selfless Service (Low)	IC: Discourage SC: Loyalty -3 Selfless Service -2 MC: Personal Courage -2
Fail	IC: Encourage MC: Personal Courage (Low) Strength (Low)	IC: Quit MC: Personal Courage -1 Strength -3 Loyalty +1
Quit	IC: Discourage	IC: Quit MC: Strength -3 Loyalty -1
Finish(2)	IC: Encourage MC: Strength (Med - Hi) or Personal Courage (Med - Hi)	IC: Finish MC: Strength +2 Loyalty +3

IC: Internal Connector, MC: Main Character, SC: Supporting Character

Table 7. Interactions for Basic Physical Training: Run Qualification Scene

2. Exemplar Story

An exemplar story is provided in Appendix A. The description begins with the player's selection of an actor and definition of a character, and continues through the initial scenes of a basic training story.

G. OBJECTS

In this application, objects are associated closely with goal achievement. There is a relatively small set of objects, relating to possessions the character can accumulate or long term commitments on the part of the character (Table 8). These possessions and commitments carry with them recurring costs, either financial or in terms of time, or both. For example, buying a car results in recurring car payments. Failure to meet the commitments can lead to difficulties. On the other hand, successfully managing the

commitments leads to rewards in terms of strengthening the character's values and resources, and exposing additional goal opportunities.

Car
Sports Utility Vehicle
Luxury car
TV
Stereo
Computer
Video Games
Home Entertainment Center
Steady Relationship
Spouse
Baby

Table 8. AA: Soldiers Story World Objects

H. SUMMARY

AA: Soldiers provided a unique opportunity to combine research with application. It was a collaborative effort involving a team of nine artists, sound technicians and programmers working on various media production and software development efforts associated with the Interactive Story Generation System. The efforts included development of the animation engine, location generator, text-to-voice system and interface, with parallel and complimentary efforts to generate the underlying media.

The *AA: Soldiers* project serves as proof-of-concept that the CMAS architecture, and story engine CMAS, described in this dissertation are capable of generating interactive stories.

THIS PAGE INTENTIONALLY LEFT BLANK

VIII. CONCLUSIONS AND RECOMMENDATIONS

This chapter summarizes the research presented in this dissertation. It begins by highlighting the major contributions, and is followed by a description of possible avenues for application. While this research made contributions to the field of interactive stories and multi-agent system simulation, it also raised a number of important questions. Accordingly, the chapter concludes with a discussion of recommendations for future work.

A. CONCLUSIONS

The story engine provides a fundamentally new approach for generating interactive stories. The underlying concept moves away from centralized control, and generates stories as a by-product of agent interactions through the distributed architecture of a multi-agent system. The story engine is a robust simulation engine that is not tied to any single domain or display medium. The domain independent nature of the story engine is a characteristic inherited from the general CMAS architecture. The bottom-up design employed by the story engine allows it to scale to large story worlds, and generate story lines from a factorially huge story space. With the protagonist-centric exploration of the story world, where the protagonist responds to the user's interventions, there is a strong sense of perceived consistency in the story lines as the user guides their character's journey.

The story engine CMAS definition provides a formal and descriptive notation for translating an abstract story world into software. Using such an architecture, researchers can explore, manipulate and stress story worlds through the use of agents.

The formal description of a Connector-based Multi-Agent System (CMAS) architecture, along with accompanying description of an agent communication, coordination and control process based on *connectors* and *connecting*, provides an avenue for researchers to investigate the capabilities of connector-based simulation.

The proof-of-concept implementation of the story engine in *AA: Soldiers* demonstrates the feasibility of the CMAS architecture, and specifically, the instance of the architecture as defined by the story engine CMAS. The specific components and

operations of the story engine CMAS capture the requisite elements to describe a story world and, more importantly, generate story lines.

B. APPLICATIONS

Returning to the initial motivation for this research, founded in the belief that there is value to be gained from "linking entertainment and defense," the applications for interactive story are resident in both domains. The story engine provides an advanced simulation methodology for training and education, as well as for story-based interactive entertainment. While the story engine blurs the lines between simulation and entertainment, this research leaned more toward defense application at the possible expense of immersive narrative entertainment.

1. Educational Gaming

The *America's Army: Soldiers* project is a game with a message. The Army recognizes that in order to attract young people to the military, it must first educate them as to what the Army has to offer. In this sense, the story engine is very much an information packaging and presentation tool. At the same time, it is important to present the message in a form that is appealing to the target audience (i.e., 18 to 24 year olds). On this front, the entertainment value of the engine takes center stage. By combining accurate information with a game-like interface, and presenting it in a personalized story format, it is possible to achieve an engaging, educational, and entertaining experience.

2. Scenario-Based Training

Scenario-based training involves the use of scenarios to help people better understand the decisions they have to make on a day-to-day basis. It is particularly effective for situations where there is no single right or wrong answer. When presented correctly, scenario-based training goes beyond what can be found in "the manuals" and challenges the trainee with events anchored in lessons learned and on-the-job experience. Allowing a trainee to explore paths that may lead to undesirable outcomes is oftentimes more valuable than simply providing the answer. Recognizing the correct path is a matter of both knowledge and experience. In its current form, the story engine, and entire ISGS, might be utilized as a general-purpose interactive scenario-based training system.

C. FUTURE WORK

1. Narrative Structure

The story engine constructs stories that follow logical cause and effect relationships, and are goal-directed. It adheres to the constraints of the domain and generates well-structured interactive story lines detailing the character's passage through the story world. The original expectations of this research were to develop a highly scalable simulation engine to meet the above objectives, while at the same time following a pronounced narrative structure. This research concluded with the first goal met, and makes progress towards the second. Chapter II described a number of narrative structures used in modern-day screenplays. These structures provide a design model for defining software narrative templates. A character agent's most favorable connection function might be modified to rate candidate connections, not only according to goal weight, but also according to their fitness with respect to a narrative template. By influencing the character agent to connect with scene agents that are not only within it's set of candidate connections, but also aligned with a narrative template, stories with a more pronounced narrative structure will be possible.

2. Potential Outcome Modeling

The CMAS architecture provides an avenue for capturing complex domains in software. It makes it possible to model a domain of interest and explore the space using goal-directed agents searching for "interesting outcomes," where an "outcome" is described as an achieved goal, plus the path taken by the agent to achieve the goal; and "interesting" is defined by a domain-specific metric. By creating agents with malevolent goals, it may be possible to explore a domain, and thereby identify and exploit unforeseen weaknesses. In addition, it is possible to recreate the path of choices to reach the goal by following the agent's sequence of connections and connectors.

3. Improved Cognitive Architecture

The character agents employed in *America's Army: Soldiers* make use of a very simple cognitive architecture comprised of personality and aptitude state variables (core values and resources respectively). This representation is adequate for modeling character behavior with respect to a game, but it is not sufficient for modeling realistic

human performance and behavior. Realist performance is a function of imperfect perception, reduced cognitive processing and behavior that is not always optimal. These imperfections are often times due to a lack of information, misperception of stimuli, or limited cognitive resources. The result is reduced, yet realistic, human performance. [Wellbrink, 2002] is exploring the use of a complex adaptive system as the basis for a cognitive architecture that models reduced human performance. The story engine will benefit from the incorporation of an improved cognitive architecture for the character agents, particularly when the agents are used to explore complex domains for potential outcomes.

4. Relations

[Roddy and Dickson, 2000] provides a detailed discussion of relations as they apply to multi-agent systems. Included in this work is the description of an architecture for managing inter-relationships among agents called RELATE. Reformulating RELATE as a connector-based architecture, and incorporating it into connector-based composite agents will provide a notable enhancement for modeling relation-centric domains, such as those involving military command structures.

5. Generalized Connecting

In the current implementation of the CMAS architecture and story engine, connections only occur across type-matched connectors. This constraint might be relaxed to allow connectors to connect based on generalization relationships (i.e., via a superclass/subclass or superinterface/subinterface hierarchy). This generalization supports object-oriented design patterns and will permit connectors to connect with "kinds" of connectors vice matching types. When moving away from modeling highly structured domains such as military career progression, it may also be beneficial to explore the use of fuzzy sets as a basis for establishing connections [Zadeh *et al.*, 1996].

6. Agent Learning

Tickets provide a means of capturing procedural knowledge. By allowing agents to augment their ticket set during a connection, procedural knowledge (i.e., experience) can be passed from one agent to the next. For example, when an agent connects with a training scene, additional tickets that capture the learning objectives of the course could

be passed to the agent. This sort of ticket passing might be used to simulate decentralized or interactive rote learning as described by [Weiss, 1999].

D. SUMMARY

This chapter presented the major contributions of this work: a general-purpose multi-agent system simulation architecture based on connectors, and a scalable simulation architecture for generating interactive stories. Significant areas for future work still remain, both in connector-based multi-agent simulation, and in interactive stories. From British novelist Mary Augusta Ward [Columbia, 1996]:

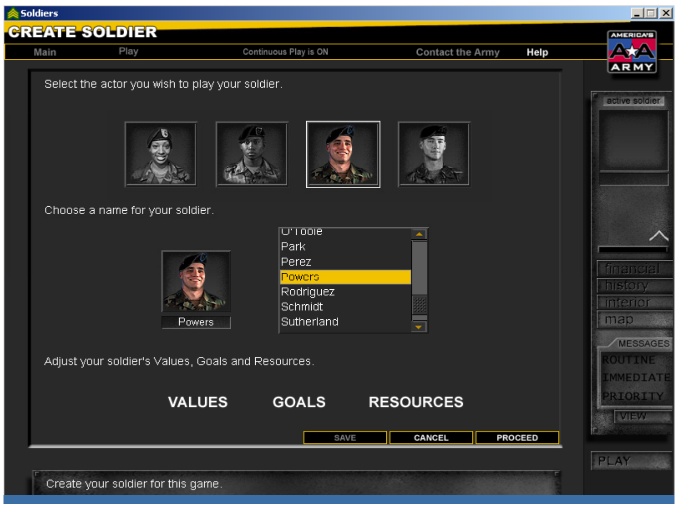
> The first law of story-telling... Every man is bound to leave a story better than he found it.

With anticipation, this research leaves both fields of study better than when they were found.

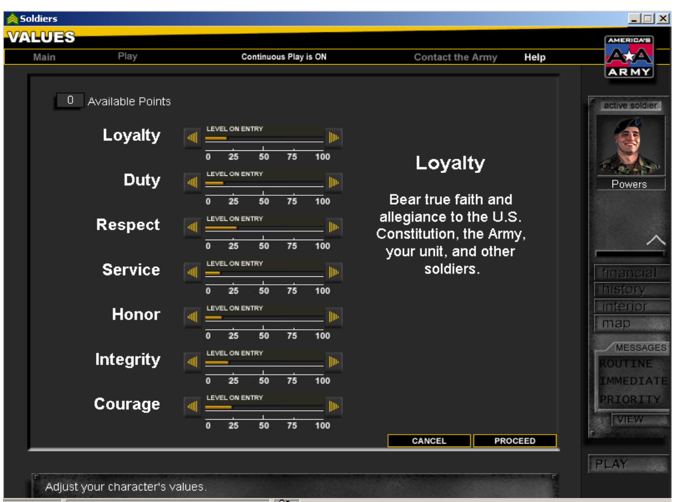
THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. EXAMPLE STORY

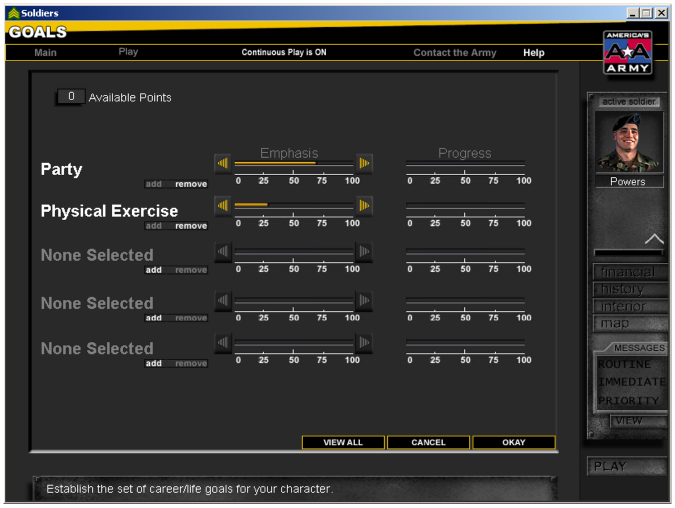
This appendix is provided to depict the initial scenes of a story session. It begins with the player defining a character and selecting a military occupational specialty (MOS). Next it presents, in chronological order, the character's arrival at basic training, initial wake-up the first morning, along with classroom and field training sessions.



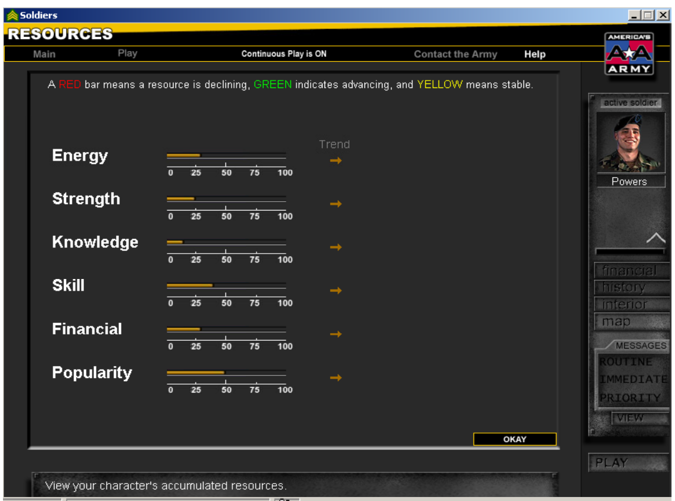
The player selects an actor and defines their character.



The initial allotment of "core value" points is distributed among the seven core values.



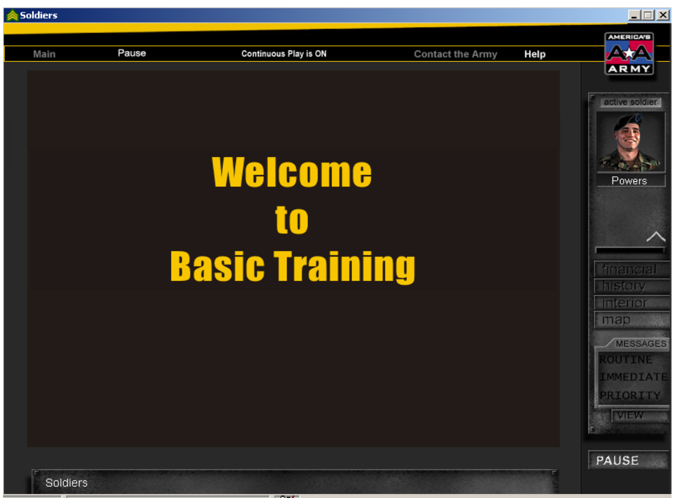
The character's initial goals are selected and prioritized.



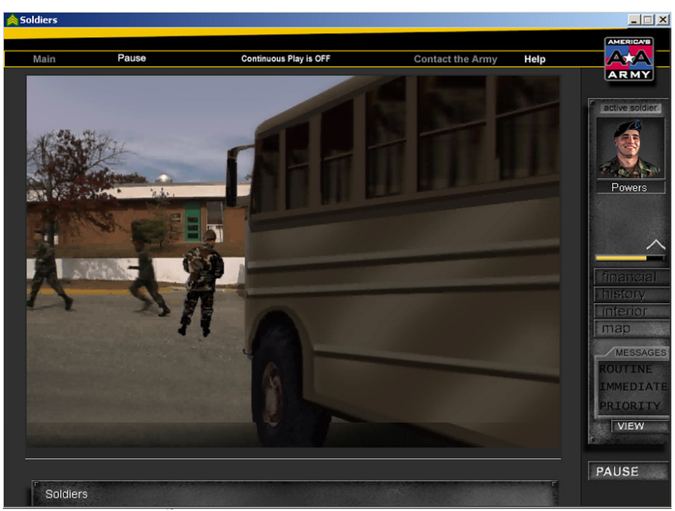
The character's resources are adjusted. This is possible at definition time only. Once the story begins, it is no longer possible to manually adjust the resources. They increase and decay based on the character's actions and achievements.

The player and character meet the recruiter.

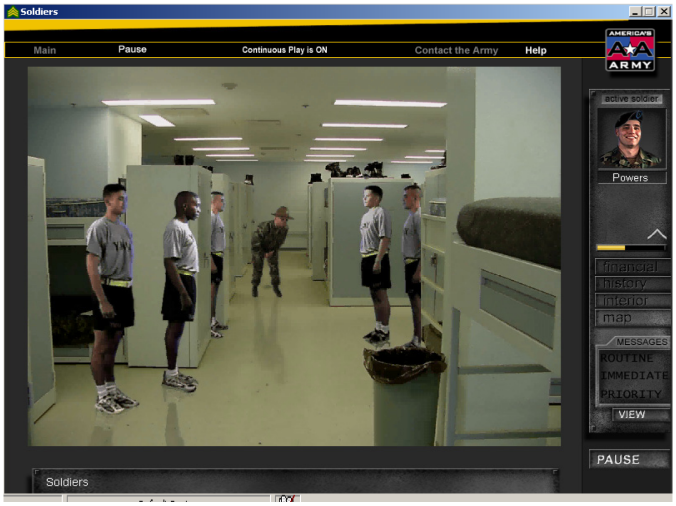
The player selects the military occupational specialty of Field Artillery for their character.



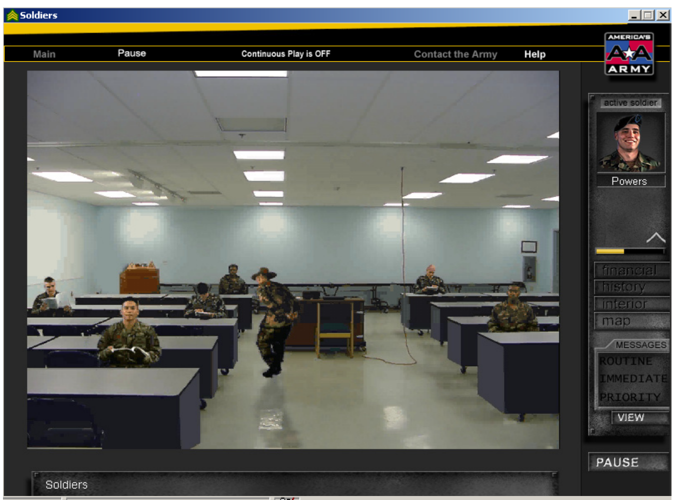
The character is sent to basic combat training (BCT).



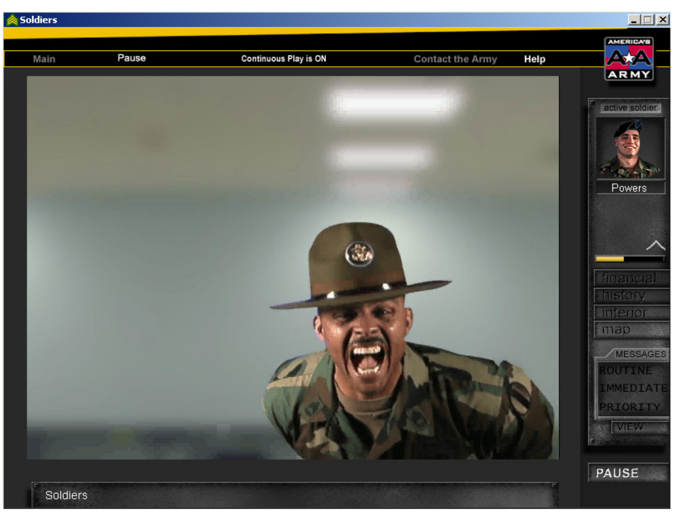
The character arrives at BCT and is welcomed by the drill instructor.



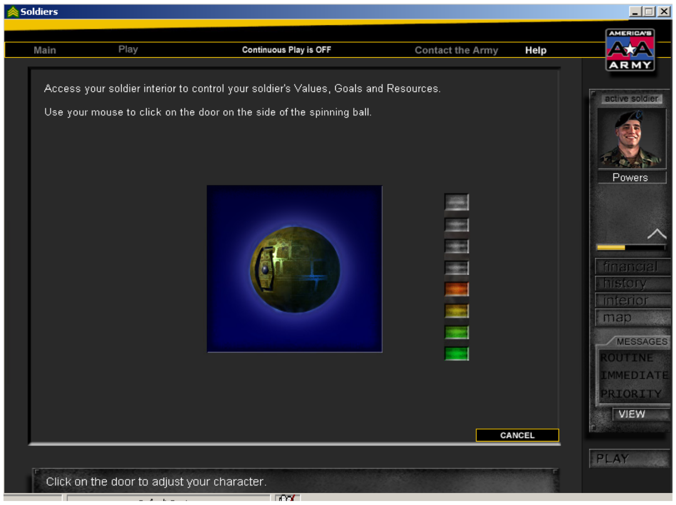
The recruits awake to the sound of a screaming drill instructor.



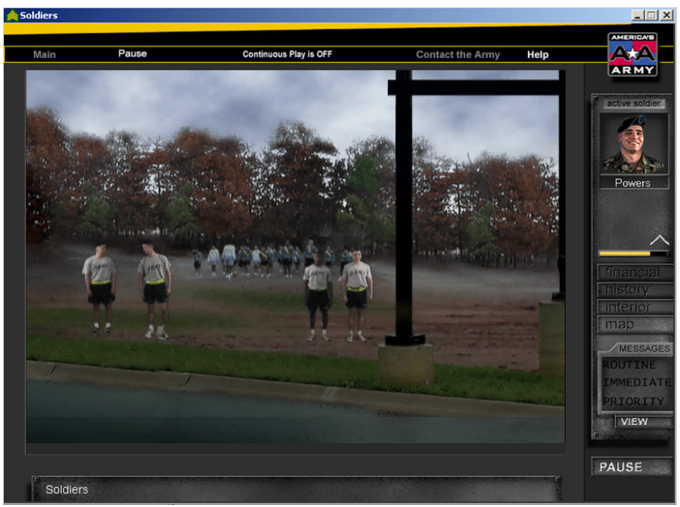
In this classroom training scene, the player's character has low energy resources and is falling asleep during the drill instructor's lecture concerning the general orders of a sentry.



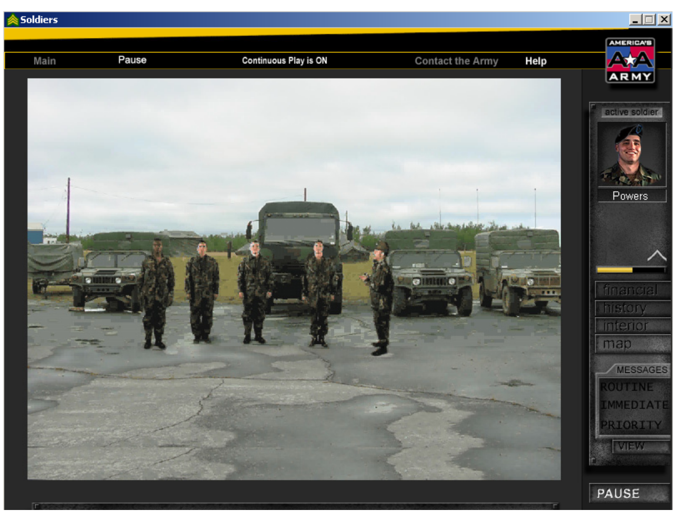
The drill instructor wakes the player's character, and tells him to stand in the back of the classroom in order to stay awake.



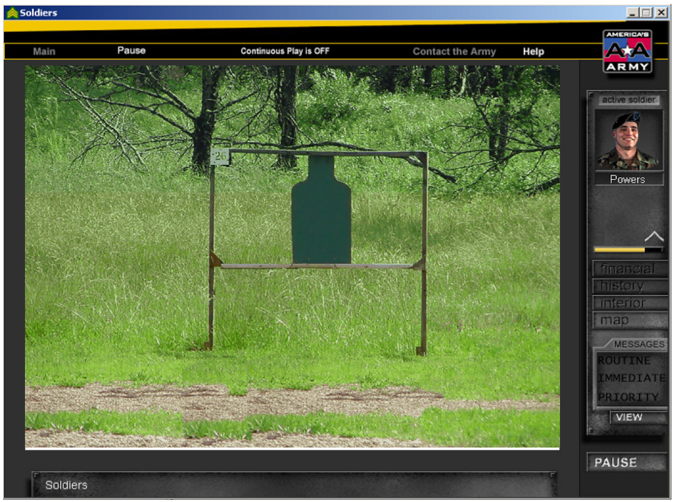
The player must adjusts their character's goals and values in order to encourage the character to focus on their low resources. The player tries to gain access to the character by clicking on the door of a quickly spinning ball. As the player gains their character's trust, the ball begins to slow down.



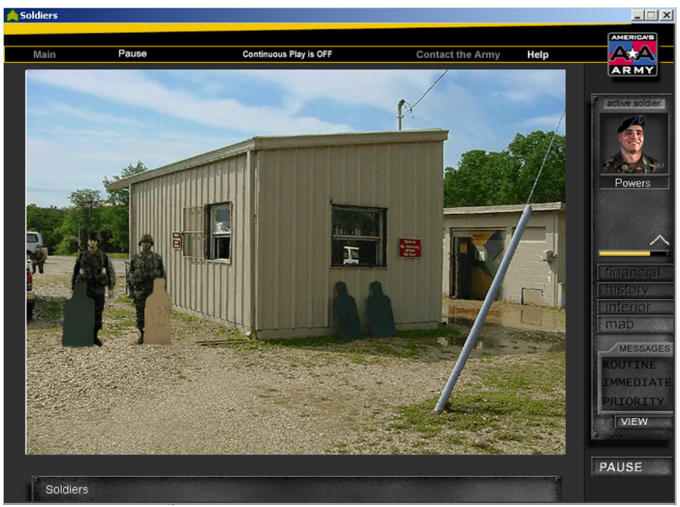
The character's goal of physical exercise will help to increase his strength and energy.



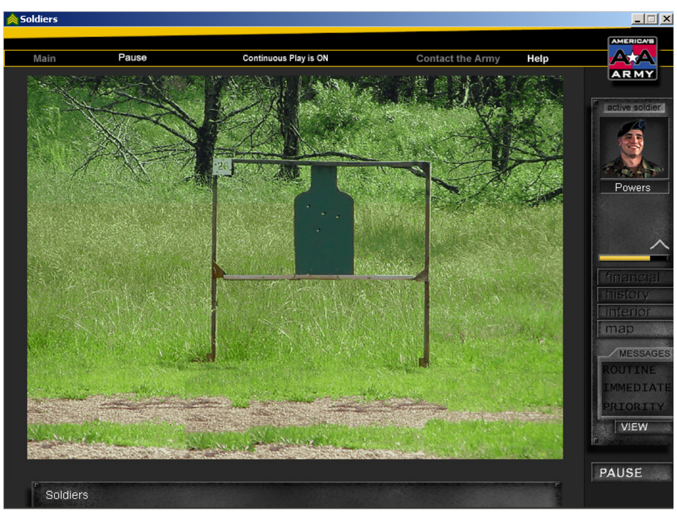
The recruits are complimented by their drill instructor for successfully completing an assigned task.



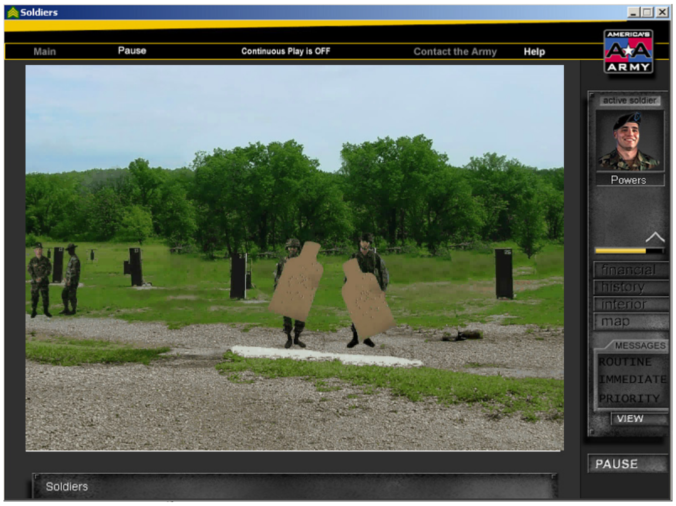
The character's initial attempt to qualify at the rifle range does not go well. He needs to pay closer attention to detail.



The character gets some encouragement and advice from his buddy before the next attempt to qualify.



The character successfully qualifies on his second attempt.



The recruits proudly display their targets to the drill instructor.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. ACRONYMS AND ABBREVIATIONS

AA: Ops	America's Army: Operations
AA: Soldiers	America's Army: Soldiers
ABL	A Behavioral Language
ABT	Active Behavior Tree
AI	Artificial Intelligence
AIT	Advanced Individual Training
A-Life	Artificial Life
BCT	Basic Combat Training
CA	Composite Agent
CARTE	Center for Advanced Research in Technology for Education at USC
CMAS	Connector-based Multi-Agent System
DAI	Distributed Artificial Intelligence
DoD	Department of Defense
E_{inner}	Inner Environment
E_{outer}	Outer Environment
ICT	Institute for Creative Technologies at USC
ISGS	Interactive Story Generation System
MAS	Multi-Agent System
MEPS	Military Entrance Processing Station
MFC	Most Favorable Connection
MOS	Military Occupational Specialty
MRE	Mission Rehearsal Exercise
MUD	Multi-User Dungeon
MVC	Model-View-Controller
NRC	National Research Council
OSUT	One Station Unit Training
PAM	Plan Applier Mechanism
PT	Physical Training
RA	Reactive Agent
SAM	Script Applier Mechanism
SCA	Symbolic Constructor Agent
SRS	Scene Rendering Subsystem
UML	Unified Modeling Language
USC	University of Southern California
VTP	Virtual Theater Project at Stanford University

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[Alberts *et al*., 2002] Alberts, B., Johnson, A. Lewis, J., Raff, M., Roberts, K and Walter, P. *Molecular Biology of the Cell*. Garland Science, New York, NY, 2002.

[Andrade, 2000] Andrade, S. F., *An Intelligent Agent Simulation of Shipboard Damage Control*, M. S. thesis, Operation Research Department, Naval Postgraduate School, June 2000.

[Aristotle, 330 BC] Aristotle. *Poetics*. Hackett Publishing Company, Indianapolis Publishing Company, Indianapolis / Cambridge, 1987.

[Bates, 1992] Bates, Joseph. Virtual Reality, Art and Entertainment. *PRESENCE: Teleoperators and Virtual Environments*, 1(1):133-138, 1992.

[Bates and Kantrowitz, 1992] Mark Kantrowitz and Joseph Bates. *Integrated Natural Language Generation Systems,* Technical Report CMU-CS-92-107, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, April 1992.

[Bates, Loyall and Reilly, 1992] Bates, J., Loyall, A., and Reilly, W. Integrating Reactivity, Goals, and Emotion in a Broad Agent. *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, Bloomington, Indiana, July 1992.

[Blumberg, 1996] Blumberg, B. Old Tricks, New Dogs: Ethology and Interactive Creatures. Ph.D. thesis, Massachusetts Institute of Technology Media Lab, 1996.

[Brooks, 1999] K. Brooks, *Metalinear Cinematic Narrative: Theory Process, and Tool*, Ph.D. thesis, MIT, Media Laboratory, Cambridge, MA, 1999.

[Columbia, 1996] *The Columbia World of Quotations*. Columbia University Press, New York, NY, 1996.

[Cooper, 1997] Cooper, G. *The Cell, A Molecular Approach*. ASM Press, Washington, DC, 1997.

[Crawford, 1999] Crawford, Chris. Assumptions underlying the Erasmatron interactive storytelling engine. Narrative Intelligence, Papers from the 1999 AAAI Fall Symposium. Technical Report FS-99-01, AAAI Press, Menlo Park, CA.

[Davenport *et al*., 2000] Davenport, G., Agamanolis, S., Barry, B., Bradley, B., Brooks, K. (2000). Synergistic Storyscapes and Constructionists Cinematic Sharing. *IBM Systems Journal*, Vol. 39, Nos. 3 & 4, 2000.

[Davis and Travers, 1997] Marc Davis and Michael Travers, 1997. A Brief Overview of the Narrative Intelligence Reading Group. *Narrative Intelligence, Papers from the 1999 AAAI Fall Symposium*. Technical Report FS-99-01, AAAI Press, Menlo Park, CA.

[Ebert, 1999] Ebert, Roger. Chicago Sun-Times, Inc, 1999. http://www.suntimes.com/ebert/ebert_reviews/1999/04/043003.html

[Echo, 2000] Echo, *John Holland's Echo*, 2000, http://www.santafe.edu/projects/echo, 30 July 2000.

[Elzenga, 2001] Elzenga, N. *The Recruits – Media Architecture System Description*. Army Game Project working document, MOVES Institute, Monterey, CA, February, 2001.

[Ferber, 1999] Ferber, J. *Multi-Agent System, An Introduction to Distributed Artificial Intelligence*. Addison-Wesley Publishers, 1999.

[Freytag, 1900] Freytag, G. *Freytag's Technique of the Drama: An Exposition of Dramatic Composition and Art*. Scott, Foresman, Chicago, 1989.

[Galyean, 1995] Galyean, Tinsley A III. Narrative Guidance of Interactivity. Ph.D. thesis, School of Architecture and Planning, Massachusetts Institute of Technology, June 1995.

[Gamma *et al*., 1995] Gamma, E., Helm, R., Johnson, R., Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[Hayes-Roth *et al*., 1995] Hayes-Roth, B., Brownston, L., Sincoff, E. *Directed Improvisation by Computer Characters.* Technical Report KSL-95-04. Knowledge Systems Laboratory, Department of Computer Science, Stanford University, 1995.

[Hayes-Roth *et al*., 1996] Hayes-Roth, B., van Gent, R. and Huber, D. *Acting in Character*. Technical Report KSL-96-13. Knowledge Systems Laboratory, Department of Computer Science, Stanford University, 1996.

[Hayes-Roth and Rousseau, 1997] Hayes-Roth, B. and Rousseau, D. *Improvisational Synthetic Actors with Flexible Personalities.* Technical Report KSL-97-10. Knowledge Systems Laboratory, Department of Computer Science, Stanford University, 1997.

[Hiles *et al*., 2001] Hiles, J., VanPutte, M., Osborn, B., Zyda, M., *Innovations in Computer Generated Autonomy at the MOVES Institute*, Technical Report NPS-MV-02-002, Naval Postgraduate School, Monterey, California, 2001.

[Holland, 1995] Holland, J. *Hidden Order.* Perseus Books, Reading, Massachusetts, 1995.

[ICT, 2002] Institute for Creative Technologies, University of Southern California, 2002. http://www.ict.usc.edu/disp.php

[Ilachinski, 1997] Ilachinski, A. *Irreducible Semi-Autonomous Adaptive Combat (ISAAC): An Artificial-Life Approach to Land Warfare*, Center for Naval Analysis Research Memorandum CRM 97-61.10 August, 1997, Center for Naval Analysis, Alexandria, VA, 1997.

[Kelso *et al.*, 1993] Kelso, Margaret T., Weyhrauch, Peter, Bates, Joseph. Dramatic Presence. *PRESENCE: Teleoperators and Virtual Environments*, 2(1), Winter 1993.

[Kessler and Kilgore, 1997] Kessler, M and Kilgore, R. Bringing an Agent to Life. In *Proceeding of Workshop on Animated Interface Agents: Making them Intelligent.* International Joint Conference on Artificial Intelligence, 1997.

[Langton, 1997] Langton, C. (Ed), *Artificial Life: An Overview*, The MIT Press, Cambridge, MA, 1997.

[Laurel, 1986] Toward the Design of a Computer-Based Interactive Fantasy System. Ph.D. thesis, Drama Department, Ohio State University, 1986.

[Laurel, 1991] Laurel, Brenda, 1991. *Computers as Theater*. Addison-Wesley, Reading, MA, 1991.

[Lebling *et al*., 1979] Lebling, D., Black, M., Anderson, T. *IEEE Computer*, Vol. 2, No. 4, April 1979.

[Loyall and Bates, 1991] Loyall, A. and Bates, J. Hap: A Reactive Adaptive Architecture for Agents. Technical Report CMU-CS-91-147. Department of Computer Science. Carnegie Mellon University.

[Loyall, 1997] Loyall, A. Bryan. Believable Agents: Building Interactive Personalities. Ph.D. thesis, Technical Report CMU-CS-97-123, School of Computer Science, Carnegie Mellon University, 1997.

[Marsella *et al.*, 2000] Marsella, S., Johnson, W., LaBore, C. *Interactive Pedagogical Drama*. Papers from Agents 2000, Barcelona, Spain, 2000.

[Mateas, 1997] Mateas, Michael. *An Oz-Centric Review of Interactive Drama and Believable Agents*. Technical report CMU-CS-97-156, Carnegie Mellon University, Department of Computer Science, June 1997.

[Mateas and Stern, 2000] Mateas, M and Stern, A. Towards Integrating Plot and Character for Interactive Drama. In *Proceeding of Socially Intelligent Agents: The Human in the Loop.* AAAI symposium, Nov 2000.

[Mateas and Stern, 2002] Mateas, M and Stern, A. A Behavioral language for Story-based Believable Agents. In *Working notes of Artificial Intelligence and Interactive Entertainment*. AAAI Spring Symposium Series. Menlo Park, CA. AAAI Press, 2002.

[Meehan, 1976] Meehan, J. The Metanovel: Writing Stories by Computer. Ph.D. thesis, Yale University, 1976.

[Merriam-Webster, 2002] *Merriam-Webster's Collegiate Dictionary*, Tenth Edition copyright 2001 by Merriam-Webster, Incorporated.

[Murray, 1998] Murray, Janet, 1998. *Hamlet on the Holodeck*. Cambridge, MA: MIT Press, 1998.

[NRC, 1997] Zyda, Michael and Sheehan, Jerry (eds.). *Modeling and Simulation: Linking Entertainment & Defense*, National Academy Press, September 1997.

[Oshuga *et al*., 1997] Oshuga, A., Nagai, Y., Irie, Y., Hattori, M., and Honiden, S. PLANGENT: An Approach to Making Mobile Agents Intelligent, *IEEE Internet Computing*, Vol. 1, No. 4, 1997.

[Pausch *et al.*, 1996] Pausch, R., Snoddy, J., Taylor, R., Watson, S., Haseltine, E. Disney's Aladdin: First Steps Toward Storytelling in Virtual Reality. In *Computer Graphics Proceedings, Annual Conference Series*, 1996.

[Polti, 1977] Polti, Georges. *The Thirty-Six Dramatic Situations*. The Writer, Inc, 1977.

[Reilly, 1996] W. Scott Neal Reilly. Ph.D. thesis, Believable Social and Emotional Agents. Technical Report CMU-CS-96-138, School of Computer Science, Carnegie Mellon University, 1996.

[Rickel *et al*., 2001] Rickel, J. Gratch, R. Hill, S. Marsella, and W. Swartout, Steve Goes to Bosnia: Towards a New Generation of Virtual Humans for Interactive Experiences. In *AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*, Stanford University, CA, March 2001.

[Roddy and Dickson, 2000] Roddy, K and Dickson, M. Modeling Human and Organizational Behavior Using a Relation-Centric Multi-Agent System Design Paradigm. Masters thesis, Naval Postgraduate School, 2000.

[Rowe *et al*., 2000] Rowe, N., Andrade, S., Gaver, D., Jacobs, P. Analysis of Shipboard Firefighting-Team Efficiency Using Intelligent-Agent Simulation. In *Proceedings from Command & Control Research & Technology Symposium, 2002*. Naval Postgraduate School, Monterey, CA, June, 2002.

[Russell and Norvig, 1995] Russell, S. and Norvig, P. *Artificial Intelligence, A Modern Approach*, Prentice Hall, 1995.

[Schank and Abelson, 1977] Schank, Roger C. and Abelson, Robert P. 1977. *Scripts, Plans, Goals and Understanding*. Lawrence Erlbaum Associates, Publishers, 1977.

[Schank, 1990] Schank, Roger, 1990. *Tell me a story: A new look at real and artificial memory*. New York: Scribner, 1990.

[Scieszka and Smith, 1989] Scieszka, J., Smith, L. *The True Story of the 3 Little Pigs*. Viking Kestrel, New York, N.Y., 1989.

[Siegel, 2001] Siegel, David, *The Nine-Act Structure*, www.dsiegel.com. January 2001.

[Szilas, 2001] Szilas, Nicolas, A new Approach to Interactive Drama: From Intelligent Characters to an Intelligent Narrator. AAAI Spring Symposium on AI and Interactive Entertainment, 2001.

[Swartout *et al.*, 2001] Swartout, W., Hill, R., Gratch, J., Johnson, W.L., Kyriakakis, C., LaBore, C., Lindheim, R., Marsella, S., Miraglia, D., Moore, B., Morie, J., Rickel, J., Thiebaux, M., Tuch, L., Whitney, R., Douglas, J. Toward the Holodeck: Integrating Graphics, Sound, Character and Story. In *Proceeding of the Fifth International Conference on Autonomous Agents*, pp. 409-416, May 2001. New York, ACM Press.

[Thompson, 1999] Thompson, Kristin. *Storytelling in the New Hollywood: Understanding Classical Narrative Technique*. Harvard University Press, Cambridge, Massachusetts, 1999.

[U.S. Army, 1999] Department of the Army. *Army Leadership, Be, Know, Do*, Field Manual No. 22-100. Headquarters, Department of the Army, Washington, DC, 1999.

[VanPutte *et al.*, 2001] VanPutte, M., Osborn, B., Hiles, J. A Composite Agent Architecture for Multi-Agent Simulations, *Proceedings of the Eleventh Conference in Computer Generated Forces and Behavior Representation*, May 2002.

[VanPutte, 2002] VanPutte, M. A Computational Model and Multi-Agent Simulation for Information Assurance. Ph.D. thesis. Department of Computer Science, Naval Postgraduate School, Monterey, CA, 2002.

[Voytilla, 1999] Voytilla, Stuart. *Myths and the Movies: Discovering the Mythic Structure of 50 Unforgettable Films*. Michael Wiese Productions, 1999.

[Weiss, 1999] Weiss, G. (ed). *Multiagent Systems, A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999.

[Wellbrink, 2002] Wellbrink, J. A Reduced Human Performance Model for Exploring Unintended Consequences and Potential Outcomes. Ph.D. thesis proposal. The MOVES Institute, Naval Postgraduate School, Monterey, CA, 2002.

[Weyhrauch, 1997] Weyhrauch, P. Guiding Interactive Drama. Ph.D. thesis, Technical Report CMU-CS-97-109, School of Computer Science, Carnegie Mellon University, 1997.

[Wooldridge, 2002] Woodridge, M. *An Introduction to MultiAgent Systems*. John Wiley & Sons Ltd, West Sussex, England, 2002.

[Young, Pollack, & Moore, 1994] Young, R. M., Pollack, M. E., and Moore, J. D.. Decomposition and Causality in Partial Order Planning. In *Proceedings of the Second International Conference on AI and Planning Systems*, 1994.

[Young, 1999] Young, R. Notes on the use of Plan Structures in the Creation of Interactive Plot. In *Narrative Intelligence, Papers from the 1999 AAAI Fall Symposium*. Technical Report FS-99-01, AAAI Press, Menlo Park, CA.

[Young, 2000] Young, R. Creating Interactive Narrative Structures: The Potential for AI Approaches. In *The Working Notes of the AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*, Stanford, CA. AAAI Press, March 2000.

[Zadeh *et al.*, 1996] Zadeh, L., Klir, G., Yuan, B. *Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems: Selected Papers by Lotfi A. Zedeh (Advances in Fuzzy Systems - Applications and Theory , Vol 6)*. World Scientific Publishing Co, River Edge, NJ, 1996.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, VA

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, CA

3. Dr. Michael Zyda
   Director, MOVES Institute
   Naval Postgraduate School
   Monterey, CA

4. Dr. Ted Lewis
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA

5. Dr. Don Brutzman
   Department of Applied Science
   Naval Postgraduate School
   Monterey, CA

6. Dr. Rudy Darken
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA

7. Dr. Michael Capps
   Epic Games
   Raleigh, NC

8. Professor John Hiles
   MOVES Institute
   Naval Postgraduate School
   Monterey, CA

9. Commander Brian Osborn
   MOVES Institute
   Naval Postgraduate School
   Monterey, CA

10. CAPT Michael Lilienthal
    Director, Defense Modeling and Simulation Office
    Alexandria, VA

11. CAPT Roland Mulligan
    N6M
    2000 Navy Pentagon
    Washington, DC

12. LTC Casey Wardynski
    Director, Office of Economic and Manpower Assessment
    United States Military Academy
    West Point, NY

13. Jim Weatherly
    N60M
    2000 Navy Pentagon
    Washington, DC